Morphflow : Une approche générique basée sur les opérations pour la gestion du changement dynamique dans les Workflows flexibles

Karim Dahmen

Nancy-Université
Institut National Polytechnique de Lorraine

Institut National Polytechnique de Lorraine

Morphflow : Une approche générique basée sur les opérations pour la gestion du changement dynamique dans les Workflows flexibles

MÉMOIRE

soutenu le 25 juin 2008

pour l'obtention du

Master Recherche de l'Institut National Polytechnique de Lorraine

(Mention Informatique - Spécialité Services Distribués et Réseaux de Communication)

par

Karim DAHMEN

Composition du jury :

	Mme. Noëlle Carbonell	Professeur UHP-Nancy1
	M. Didier Galmiche	Professeur UHP-Nancy1
	M. Claude Godart	Professeur UHP-Nancy1 / ESSTIN
	M. Dominique Mery	Professeur UHP-Nancy1 / ESIAL
	M. Guy Perrier	Professeur Nancy 2
Encadrant :	M. Charoy François	Maître de Conférences UHP-Nancy1 / ESIAL

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Remerciements

Mes remerciements s'adressent aux membres du jury qui ont bien voulu accepter de juger ce modeste travail.

Avec le plus grand respect, je tiens à remercier Monsieur Claude Godart de m'avoir accueilli dans son équipe me permettant de travailler avec des personnes remarquables.

Je voudrais que Monsieur François Charoy, mon maître de stage, trouve ici l'expression de mon indéfectible reconnaissance pour le soutien, la disponibilité et les conseils éclairés qu'il n'a cessé de me prodiguer tout au long de ce stage. Je vous remercie de votre confiance pour la liberté des choix que j'ai pu faire pour mener à bien ce projet. Aujourd'hui, je suis conscient de cette responsabilité et vous suis sincèrement reconnaissant de m'avoir fait découvrir tout ce que j'ai appris. J'espère que notre collaboration puisse continuer.

Je remercie également tout l'effectif de l'équipe ECOO pour leur chaleureux accueil.

Enfin, je dédie ce travail à ma mère, ma sœur pour leurs sacrifices et les marques d'amour et de tendresse dont ils ne se sont jamais départis, et à tous ceux qui ont fait que ce travail soit réalisé. Clin d'œil à ma chère Alevtina pour son soutien et sa patience.

Le souci de mon ignorance me rassure...
Mais l'assurance de mes connaissances est un constant souci.

Résumé

Un système de gestion de Workflows (*Workflow Management System* : WfMS) est un dispositif logiciel permettant de modéliser, d'automatiser et de surveiller l'exécution des flux de travail. Pour une automatisation efficiente, et mieux refléter le procédé réel, le modèle contenu dans le WfMS doit minimiser, tout en étant exempt d'erreurs, l'écart entre sa représentation et le processus réel. Pour ce faire, le déploiement du WfMS devrait éviter de reproduire la rigidité introduite par une planification rigoureuse des procédés métiers. En revanche, l'automatisation doit aussi admettre que des utilisateurs agréés puissent faire dévier les exécutions d'un Workflow de leur planification initiale, par exemple, face à une situation exceptionnelle, et faire évoluer les modèles de Workflows face aux optimisations ou à un changement des législations régissant les procédés métiers. Dans ce contexte, l'enjeu est d'offrir aux administrateurs du moteur d'exécution un support efficace pour la gestion du changement. La littérature propose une multitude de solutions *basées sur les états* d'avant et d'après le changement pour résoudre la problématique des modifications- par exemple, la cohérence et la robustesse - des Workflows en cours d'exécution. Sauf que la complexité accrue de ces solutions pose de nombreux soucis d'ingénierie lorsqu'il s'agit de les mettre en œuvre. Outre la difficulté de leurs exploitations, la traçabilité, le suivi, l'analyse et la réutilisation des changements sont considérablement limités. Dans ce mémoire, nous présentons une approche innovante basée sur des opérations pour la gestion des changements métiers (à savoir, l'évolution du schéma et la modification cohérente des instances en cours d'exécution) qui se veut être à un haut niveau de généricité sans, pour autant, affecter la robustesse des WfMS. Nous proposons pour cela un canevas formel *basé sur les opérations* et un nouveau paradigme de systèmes d'opérations- Stratégie/Incidence(s)- offrant un cadre flexible pour effectuer des changements, afin de faire évoluer les modèles de Workflow tout en préservant des exécutions correctes. Dans le but de préserver la correction des exécutions, par exemple, une incidence conformément à une stratégie, le critère de correction dynamique des opérations de changement que nous introduisons, permet de garantir (décidé localement dans un système d'opérations) qu'une Dérive (c'est-à-dire, l'incidence) puisse évoluer selon l'évolution de son Noyau (à savoir, la stratégie). Comme l'évolution des modèles ou les changements in situ des exécutions reflètent un savoir-faire et nécessitent une expertise, il est nécessaire de pouvoir capitaliser le changement en proposant des techniques pour leur réutilisation. Finalement, nous définissons un graphe des arrangements lexicographique d'opérations centré sur le noyau dont la topologie permet le suivi et la traçabilité des opérations de changement pour l'analyse et la réutilisation des modifications des modèles dans un WfMS.

Mots-clés: Morphflow, Flexibilité des Workflows, Système d'opérations, Gestion du changement, Stratégie/Incidence, Noyau/Dérive, Correction structurelle dynamique basée sur les opérations.

Abstract

A Workflow Management System (WfMS) is a software system that provides specification, enactment and monitoring of Workflows. In order to provide effective process support, WfMS should capture real-world processes adequately, *i.e.*, no mistakes should be found between the computerized processes and real ones. In order to achieve this goal, the introduction of WfMS must avoid the rigidity and the freeze's introduced by a rigourous planning of Business Processes. In the opposite, WfMS should allow authorized users to flexibly deviate from predefined processes as required (*e.g.*, to deal with exceptions) and to evolve Workflow models over time (*e.g.*, due to process optimizations or legal changes). In this context, increasing demand for process change support poses new challenges for Workflow Administrators and requires the use of change enabling techniques. Many state-based approaches promise flexible software solutions for realizing adaptive WfMS, but their complexity to cope with fundamental issues related to dynamic process changes (*e.g.*, correctness and robustness) makes their implementation very complex. This in turn makes it difficult for WfMS engineers to exploit change support capabilities of these technologies, to monitor change traceability and analysis, and finally, changes reuse. In this work, we provide a formal framework for an innovative operation-based approach enabling structural changes (*i.e.*, schema evolution and Ad-hoc changes) at a high level of abstraction and without affecting robustness of WfMS. Specifically, we introduce a novel paradigm of Operations System- Strategy and Incidence(s) - tackling problems related to change and providing a correct and flexible scope for evolving Workflow models. In order to obtain the correctness of Workflow executions (*e.g.*, an Incidence under a Strategy) an operation-based dynamic criterion which preserves an accurate evolution of a Drift (*i.e.*, the Incidence) according to its Kernel (*i.e.*, the Strategy) is introduced. Finally, for monitoring purposes, we introduce a Kernel-Centred Lexicographical Arrangements of Operations Graph as a features to support change traceability and analysis. Accordingly, as deviations (*i.e.*, models evolution and Ad-hoc changes) require significant user experience and effort to define changes from scratch, our framework offers a support for change reuse.

Keywords: Morphflow, Workflow flexibility, Change management, Operations system, Strategy/Incidence, Kernel/Drift, Dynamic operation-based structural correctness.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Selon une tendance actuelle, et pour pouvoir réagir au mieux à l'évolution de leurs besoins en traitements de l'information, les organisations contemporaines sont régulièrement amenées à utiliser un large panel de nouvelles technologies, elles-mêmes évolutives. Ce potentiel de réactivité passe impérativement par une utilisation efficiente d'outils efficaces. Ceci dit, et à une échelle réduite, il existe un nombre conséquent d'outils informatiques d'assistance (traitement de texte, tableur, *etc.*) répondant précisément aux besoins des utilisateurs pris individuellement. Mais à une échelle plus importante, celle d'un groupe de personnes, les **outils collaboratifs** (*Groupware*), même s'ils se font plus rares, ne sont pas capables de saisir toutes les règles organisationnelles [1] (*Business Rules*) régissant le groupe qu'ils assistent. Par exemple, un éditeur de texte collaboratif n'a pas de connaissance organisationnelle sur le document qui est entrain d'être édité. Ainsi, le développement d'outils collaboratifs conscients de leur environnement organisationnel amène les organisations à se doter de systèmes potentiellement plus puissants et plus utiles. Une classe particulière de ces systèmes est le ***Workflow*** (traduit littéralement « flux de travail»).

Les systèmes de Workflow permettent d'assister un ou plusieurs groupes de personnes dans la réalisation d'un **procédé métier**[2] (aussi appelé processus opérationnel ou procédure d'entreprise) en capturant la connaissance organisationnelle et en permettant de cadencer les flux de travail. Plus explicitement, le terme de « *Workflow* » désigne la « gestion électronique des procédés métiers». De façon plus pratique, le Workflow est définit comme un « système assistant une organisation dans la modélisation, l'exécution, le suivi et la coordination du flux d'accomplissement des tâches dans un environnement distribué» [1]. Ce segment logiciel particulier de la **gestion des procédés métiers** (*Business Process Management* : BPM) au sein du marché de l'intégration adresse la gestion des procédés métiers dans leur intégralité, depuis la modélisation, la conception et le design jusqu'au déploiement et l'exécution, en passant par le suivi et l'optimisation des processus faisant appel aux acteurs humains (Workflow) autant qu'aux systèmes (automatisation). Ces produits sont communément appelés systèmes de gestion de procédés (*Process Management Systems* : PMS).

Plus particulièrement, si un procédé permet de décrire l'organisation d'un groupe de personnes, les règles de travail qui les régissent et les règles organisationnelles qui les lient, un **système de gestion de procédés métiers** (*Workflow Management System* : WfMS) permet de formaliser, de structurer, d'automatiser (dans la mesure du possible) et d'exécuter ces processus. Il en résulte que ce système est constitué de deux entités logicielles distinctes. La première est celle du **modèle de Workflow** (*Workflow Model*) ou le module de spécification (*Specification Module*) qui permet aux administrateurs et aux analystes de définir les procédures métiers, de les analyser, de les simuler et de les affecter aux acteurs nécessaires. Cette entité constitue alors les objectifs du procédé, les structures de contrôle, les structures de données, les structures organisationnelles, les structures de conversation, *etc.*. Cependant, ce modèle de Workflow doit (à priori) définir les étapes du procédé métier et décrire l'ordre dans lequel elles doivent figurer pour former le flux de travail.

La seconde entité est le **moteur d'exécution** (*Workflow Execution Module* ou *Workflow Enactment*

[1] *Une règle organisationnelle, ou règle d'affaire, définit ou contraint un aspect du métier d'une organisation afin de structurer et de contrôler le comportement de son travail.*

[2] *Ensemble d'activités organisées de manière chronologique pour atteindre un objectif, généralement délivrer un produit ou un service, dans le contexte d'une organisation de travail (par exemple, une entreprise, administration, etc.).*

System). Ce dispositif logiciel exécute et coordonne une ou plusieurs définitions de procédés et représente l'interface utilisée par l'utilisateur final. Il permet généralement un suivi et une identification des acteurs en précisant leur rôle et la manière de le remplir au mieux pour satisfaire le flux de travail. Par abus de langage, on appellera ce dispositif logiciel tout simplement « Workflow». C'est dans ce contexte que les Workflows sont basés sur les cas d'exécution (*Case-Based*), c'est-à-dire, que chaque partie du travail à réaliser est exécutée pour un cas spécifique.

Par une taxonomie plus générale, un **système d'information conscient du procédé métier** (*Process-Aware Information System* : PAIS) est un système d'information particulier permettant la séparation entre la logique métier et le codage applicatif. Au moment de l'exécution (*Run-Time*), le PAIS orchestre les procédés conformément à une logique prédéfinie. Les WfMSs (par exemple, *Staffware* [2], *ADEPT* [3], *WASA* [4]) et les **systèmes de gestion d'instances** (*Case-Handling Systems*), en citant Flower [5], sont des technologies typiques d'un PAIS. Puisqu'on s'intéresse à l'aspect spécifique du changement dans ces systèmes, nous ne faisons pas la distinction, dans la suite, entre WfMS et PAIS. Mais nous les citerons, pour leur potentiel de gestion des procédés, par PMS.

Selon notre vision, nous considérons que ces deux dimensions- c'est-à-dire, la modélisation du Workflow et son exécution -sont intimement liées. Pour simple exemple, il doit être possible, au **niveau du modèle de Workflow** (*Workflow Type Level*), de changer le procédé métier, et succinctement, au **niveau opérationnel** (*Workflow Instance Level*), de changer dynamiquement les étapes du procédé quand celui-ci est en exécution. Nos considérations se basent sur l'observation que le changement est inhérent à la plupart des acteurs aussi bien organisationnels qu'individuels. Dans le contexte du travail moderne [6], les organisations qui se défendent du changement sont vouées à l'obsolescence puisqu'elles ne peuvent plus faire face à la concurrence. Ces organisations doivent effectuer des modifications structurelles fréquentes comme par exemple : recruter de nouveaux collaborateurs dans leur structure organisationnelle et suivre un constant mouvement de rationalisation industrielle, s'adapter à de nouvelles législations régissant leurs secteurs d'activité, se doter de moyens efficaces pour mieux gérer les crises, procéder à des restructurations. En somme s'adapter pour livrer une concurrence à l'échelle internationale. De façon plus générale, on désignera ces modifications structurelles par des **changements métiers** (*Business Changes*).

Si ces changements sont aussi importants pour la survie d'une organisation, c'est que la pratique de la modélisation des procédés métiers doit en être consciente, et devrait prendre en compte cette dynamique. Il est nécessaire alors, qu'un système d'information conscient du procédé ne soit plus seulement une solution pour l'automatisation et le suivi de l'exécution des flux de travail, mais plus encore, devenir un outil de support et de pilotage pour la conduite du changement pour ces organisations. Car de part sa fonction première, il représente et réalise le procédé réel selon une planification rigoureuse des étapes d'un processus en minimisant les erreurs et les défaillances. Cependant, il doit être en mesure d'offrir un support efficace pour le besoin d'adaptabilité des Workflows. Cette adaptabilité se traduit par deux aspects. Le premier est que le modèle des Workflows doit suivre l'évolution du procédé métier réel. Ceci doit permettre à une organisation, d'une manière la plus flexible, la plus rapide et avec le moins d'effort de faire évoluer son Workflow afin de l'améliorer ou de corriger une modélisation de processus déjà déployée. Le second est qu'un utilisateur agréé ou un groupe puissent dévier l'exécution d'un procédé par rapport à un modèle initial, afin de faire face à une situation exceptionnelle le plus rapidement possible. Cette déviation in situ du niveau opérationnel doit être permise en contrepartie de la réingénierie de la totalité du modèle du procédé initial, qui requiert une expertise et un savoir-faire onéreux. Cet aspect du changement dynamique dans les WfMS est primordial pour éviter des efforts supplémentaires (donc des investissements plus importants), lorsqu'il s'agit d'introduire une modification minime dans une seule exécution. Par conséquent, une gestion efficiente du changement implique le recourt *(i)* à un langage approprié pour exprimer les modifications dynamiques dans un métamodèle de Workflow particulier, et *(ii)* à des politiques du changement adéquates pour conduire l'adaptation des Workflows vers un modèle modifié, tout en garantissant leur correction.

Parce qu'il est nécessaire de changer, il est aussi nécessaire de pas changer, et de contrôler le changement. Car si la flexibilité apporte la liberté de faire évoluer la définition rigide et rigoureuse d'un Workflow, elle ne signifie pas des exécutions dépourvues de toute contrainte. Ainsi, la correction du changement est un facteur déterminant de son acceptation puisqu'il ne doit pas mettre en péril la robustesse du système

de gestion de procédés métiers. D'autre part, que les organisations acceptent de changer signifie que la flexibilité de l'exploitation d'**un système d'information conscient du changement des procédés métiers** doit leur garantir la traçabilité, le suivi et la réutilisation des modifications se produisant aussi bien au niveau du modèle de Workflows qu'au niveau opérationnel. En effet, ceci leur vaudra de s'aligner sur une stratégie orienté changement, et non plus une stratégie orienté processus.

Dans ce mémoire, nous présentons une approche innovante basée sur les opérations [3] pour la gestion des changements métiers (à savoir, l'évolution du schéma et la modification cohérente des instances en cours d'exécution) qui se veut être à un haut niveau de généricité sans, pour autant, affecter la robustesse des WfMS. Nous proposons pour cela un canevas (*framework*) formel basé sur les opérations et un nouveau paradigme de systèmes d'opérations- *Stratégie/Incidence(s)*- offrant un cadre générique flexible pour effectuer des changements, afin de faire évoluer les modèles de Workflow tout en préservant leur cohérence d'exécution. La généricité de notre *framework* s'exprime par une abstraction vis-à-vis de tout langage de modélisation de Workflow et son changement, ainsi que toute politique de changement. Dans le but de préserver la correction des exécutions, par exemple, une incidence conformément à une stratégie, le critère de correction dynamique des opérations de changement que nous introduisons, permet de garantir qu'une Dérive (c'est-à-dire, l'incidence) puisse évoluer selon l'évolution de son Noyau (à savoir, la stratégie). Comme l'évolution des modèles ou les changements in situ des exécutions reflètent un savoir-faire et nécessitent une expertise, il est nécessaire de pouvoir capitaliser le changement en proposant des techniques pour leur réutilisation. Finalement, nous définirons un graphe des arrangements lexicographique d'opérations centré sur le noyau dont la topologie permet le suivi et la traçabilité des opérations de changement pour l'analyse et la réutilisation des modifications des modèles dans un WfMS.

Grâce à une étude de la flexibilité dans les systèmes de gestion des *Workflows*, ces travaux sont une contribution pour définir une approche innovante afin de résoudre les problèmes de changements dynamiques que posent les changement sectoriels dans les systèmes gestion de procédés métiers. L'appellation que j'ai retenu pour identifier ce *framework* générique basée sur les opérations pour la gestion du changement dynamique dans les procédés métiers flexibles est « *Morphflow* » en référence à un morphisme des flots composant un Workflow. La suite de ce mémoire est organisée en quatre chapitres.

Le premier chapitre présentera le contexte du changement dynamique dans la flexibilité des procédés métiers, il dégagera alors les axes pertinents de notre projet par rapport aux besoins que nous avons identifiés.

Le second chapitre est consacré à l'étude théorique du contexte de la flexibilité des Workflows. En passant en revue les spécificités introduites par les changements, il situera alors notre problématique par rapport aux travaux existants. D'abord, nous orientons nos propos vers une typologie des procédés métiers pour mettre en évidence le besoin d'adaptabilité des systèmes de gestion de Workflows. Ensuite, nous nous intéresserons à la gestion du changement comme une solution pour la flexibilité des Workflows. Enfin, nous dégagerons la notion d'adaptabilité des Workflows, et nous motiverons la nécessité de recourir à des métamodèles permettant un contrôle accru de leurs exécutions flexibles.

Le troisième chapitre de ce mémoire présente une synthèse des contributions de notre travail. Nous y retracerons les éléments ayant motivé nos choix pour la démarche de réalisation que nous avons suivi dans la construction de notre *framework* générique pour le changement, et nous dégageons les aspects innovants de notre projet. A la suite, nous détaillerons le modèle de notre *framework* par une formalisation exhaustive de tous ses concepts-clés, et nous aborderons des aspects plus concrets, liés à des problématiques opérationnelles. En dernier lieu, nous évaluerons la complexité de notre solution.

Le mémoire s'achèvera alors par un rappel nos réalisations essentielles, et présentera les perspectives ouvertes par notre proposition.

[3] *La modification d'un Workflow peut être représentée par une suite d'opérations atomiques (voir la section 1.1.4) constituant des insertions et des suppressions des composants du modèle (des activités, des données, etc.).*

Chapitre 1

Cadre de référence et problématique

Sommaire

Dans ce chapitre nous présentons le contexte de la flexibilité dans les systèmes de gestion des procédés métiers. Par une étude du changement dans les modèles Workflows, nous dégagerons les axes pertinents de notre projet par rapport aux besoins que nous identifierons. Nous commençons, d'abord, par décrire le cadre de référence de notre travail et montrer la nécessité de gérer le changement dynamique dans les Workflows ainsi que les problèmes qu'il introduit. Ensuite, nous décrivons, d'une manière détaillée, la problématique de notre sujet tout en soulignant l'intérêt de la gestion du changement dans la flexibilité des WfMS. Enfin, nous motiverons les objectifs de nos réalisations.

1.1 Cadre de référence

Avant d'aborder la notion du changement dans les procédés métiers, il est nécessaire d'introduire les propriétés structurelles des modèles de Workflows. La section suivante présente les détails du métamodèle que nous entendons utiliser.

1.1.1 Modélisation des Workflows

Un modèle de Workflow est un graphe de nœuds représentant les étapes d'exécution reliées par des arcs représentant le **flot de contrôle** et le **flot de données** entre ces étapes. La plupart des modèles de Workflow, sinon tous, représentent la structure du procédé métier au travers de graphes plus ou moins spécialisés [7, 6, 8, 9, 10, 4, 11, 12, 13]. Les nœuds du graphe représentent les activités relatives aux tâches du procédé, et les arêtes représentent les flux ou l'ordre des tâches impliquées dans le processus. Les notations graphiques du modèle de Workflow permettent d'exprimer diverses configurations telles

que la séquence, la concurrence, le choix multiple et d'autres constructions [14, 15, 16, 17, 18, 19]. Ces configurations décrivent les dépendances du flot de contrôle entre les différentes activités.

L'aspect structurel du procédé métier constitue la plus grande et la plus importante portion du modèle de Workflow. Plusieurs langages de modélisation ont vues le jour, tant dans des projets de recherche, ou dans des produits commerciaux. Divers propriétés tels que **l'expressivité** et **l'exhaustivité**, sont des facteurs déterminants dans la conception de ces langages. Cependant, une définition précise des propriétés des concepts supportés par le langage, ainsi que ses critères de correction (se reporter la section 2.2), sont fondamentales pour la réussite de l'emploi générique de ce même langage. Le métamodèle proposé par la WfMC [4] distingue les composants décrits ci-dessous [20] :

- **Schéma du Workflow** : un schéma du Workflow (*Workflow Schema* ou *Workflow Schemata*) ou schéma du procédé est une représentation possible d'un procédé sous une forme permettant son automatisation (par conséquent, sa modélisation et son exécution) par un système de gestion de procédés. Il est constitué d'un réseau d'étapes, appelées « activités », modélisant les transitions entre elles, et l'ordre (c'est-à-dire, une relation de « précédence » entre les activités) de leurs occurrences ;
- **Activité** : une activité est une étape atomique dans un procédé. Chaque activité a un nom, un type, des conditions de démarrages à défaut desquelles elle ne peut s'exécuter (dites **préconditions**) et des conditions qui doivent êtres satisfaites à la terminaison de son exécution (dites **postconditions**), et des contraintes d'ordonnancement (c'est-à-dire, des contraintes d'interdépendances entre activités distinctes). Dans notre contexte, un schéma de Workflow se doit d'avoir une seule activité initiale (marquant le début du procédé) et une seule activité finale (exprimant la terminaison du procédé) ;
- **Flot de contrôle** : un modèle du procédé décrit chaque unité de travail, mais aussi la séquence, selon laquelle ce travail est réalisé pour atteindre l'objectif escompté. Un flot de contrôle spécifie donc la séquence d'exécution (c'est-à-dire, l'ordre de « précédence ») des activités au travers de la définition des **connecteurs** de contrôle entre les activités ;
- **Données d'un Workflow** : à chaque Workflow correspond un ensemble de données contenues dans des **conteneurs** et qui sont nécessaires à son exécution. Ces données incluent *(i)* des données requises en entrée des activités, *(ii)* des données pour l'évaluation des conditions et *(iii)* des données devant être échangées entre les activités ;
- **Flot de données** : un flot de données est matérialisé par des connecteurs de données entre les activités mettant en œuvre une série de correspondance entre des conteneurs de données en sortie et des conteneurs des données en entrée de chacune des activités permettant l'échange d'information entre elles ;
- **Conditions** : les conditions permettent de définir le comportement du procédé vis-à-vis de l'occurrence de certains événements. Il y a trois types de conditions. Des conditions sur les transitions associées aux connecteurs de contrôle spécifiant quand un connecteur est évalué à vrai ou à faux. Des conditions de démarrage sur une activité (par exemple, quand tous ses connecteurs de contrôle entrants seront évalués à vrai, ou quand un d'entre eux sera évalué à vrai). Les conditions de sortie sont associées à l'événement correspondant à la terminaison de l'activité.

Pour êtres plus concis, nous ignorons le reste des composants importants à savoir les rôles, les agents, les bases de données, les ressources, *etc.*, car le changement et l'adaptation de ces aspects sont hors de portée de ce mémoire. La figure 1.1 donne un exemple de notations graphiques d'un Workflow. Ce langage de modélisation [21] est conforme au standard de la WfMC, et nous allons l'utiliser pour appuyer les divers exemples dans les sections ultérieures. La figure A.2 de l'annexe A montre un exemple simplifié, basé sur ce langage, d'un procédé de traitement des demandes d'immigration pour un service consulaire.

Pour que le procédé soit accompli, il doit être exécuté par le moteur d'exécution qui gère la contingence du procédé lui même par la direction de l'évolution du cycle de vie des **instances de procédé** (ou instances du Workflow). Une instance de procédé peut être vue comme une suite d'activités planifiées. Ces activités, exécutées ou interprétées (selon le modèle de Workflow), évoluent au sein du moteur d'exécution.

[4] *Workflow Management Coalition* : La coalition de gestion de Workflows WfMC est un groupe de compagnies qui se sont jointes ensemble pour définir des normes pour permettre l'interopérabilité et l'interconnexion de différents produits de gestion de Workflows.

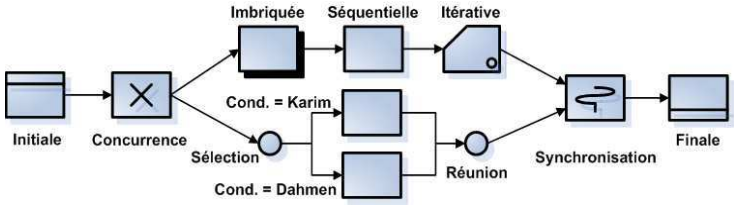


FIG. 1.1 – Notations graphiques des patrons de flot de contrôle d'un modèle de Workflow.

En effet, cette exécution peut être décrite d'une manière impérative, par exemple, comme une suite d'instructions à réaliser, comme elle peut être décrite d'une manière déclarative par des préconditions et des postconditions.

1.1.2 Qu'est-ce que le changement ?

Pour qu'un modèle de procédé puisse être exécuté, il doit être défini par un schéma de procédé S. Comme cité plus haut, un schéma est représenté par graphe orienté qui décrit un ensemble d'activités- les étapes du procédé - et les connections de contrôle entre elles. Les activités peuvent aussi bien être atomiques que contenir un sous-procédé (ou sous-Workflow, c'est-à-dire, une référence à un autre schéma S') permettant la décomposition hiérarchique d'un schéma de procédé. Par exemple, dans la figure 1.2, le schéma S^1 est composé de cinq activités : l'activité 1 peut être soit suivie par l'activité 2 parallèlement à l'activité 3 (dans un souci de brièveté, on s'intéresse peu à l'ambiguïté que peut poser une telle énonciation par rapport à la sémantique du langage utilisé [16, 14], c'est-à-dire, que la numérotation des activités ne sert qu'à les différencier, et n'évoque en rien leur ordre d'exécution). Les activités $1, 2, 3, 5$ sont atomiques, alors que l'activité 4 constitue un sous procédé ayant son propre schéma S^2. Pour les besoins de l'exécution, de nouvelles instances $I_1, ..., I_n$ sont créées et exécutées conformément au schéma S. Dans le cas de l'instance I_2 de la figure 1.2, par exemple, l'activité 1 est terminée, et l'activité 3 est activée (c'est-à-dire, offerte dans la liste des tâches de l'utilisateur [20]). Dans l'instance I_n, l'exécution en cours de l'activité B de l'instance $I_1^{S^2}$ du sous-procédé S^2 implique que l'activité 4 est aussi exécutée.

Généralement, les changements auxquelles doit faire face un PMS peuvent survenir et être réalisés à deux niveaux : au niveau du schéma du procédé ou au niveau des instances (voir figure 1.3) [12]. Pour faire face à la nature évolutive des procédés du monde réel, les changements au niveau du schéma, dits « **changements d'évolution** » (*Evolutionary Changes*) [7], deviennent nécessaires (par exemple, pour s'adapter à une nouvelle législation). D'autre part, des changements ponctuels au niveau d'une instance, dits « **changements *ad-hoc*** » ou in situ (*Ad-hoc Changes*) [12], doivent souvent êtres réalisés pour répondre à des exceptions, résultants en un schéma spécifique à l'instance. Un exemple concret pourrait être un modèle de Workflow pour la gestion des demandes d'admissions dans une université. D'un point de vue conceptuel, un tel modèle serait destiné à réaliser tous les objectifs administratifs du processus d'admission tout en maximisant son efficacité. Les instances du Workflow sont des occurrences particulières du procédé. Par analogie, l'examen du dossier d'admission d'un étudiant particulier pourrait constitué une instance du Workflow d'admission. Différentes instances d'un même Workflow peuvent représenter l'exécutions de différents sous-ensembles de tâches du Workflow.

S'agissant de changements métiers, une distinction capitale doit souligner la manière dont les répercutions de ces modifications sont reportées sur les systèmes de Workflow sous-jacents : doivent-elles affecter le modèle de Workflow, certaines instances, ou même les deux ?

Les transformations apportées au modèle de Workflow indiquent un changement permanent des procédés métiers à la suite d'optimisations du processus [11, 22], d'innovations ou de la réingénierie des processus d'affaires, ou tout simplement à cause d'erreurs avérées et survenues lors de la conception du modèle. En revanche, les changements n'affectant qu'une ou plusieurs instances représentent des transformations inopinées, probablement de plus rares situations dans les processus. Ces instances, définies par

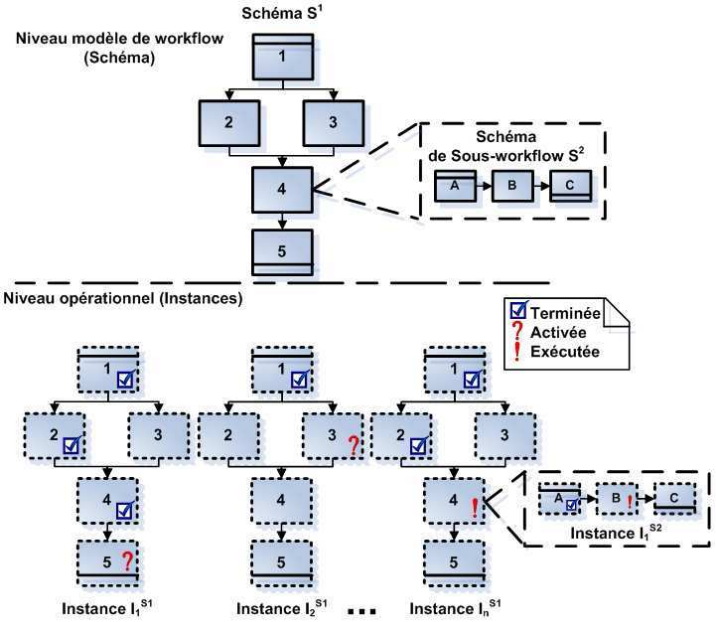


FIG. 1.2 – Illustration d'un schéma et des instances d'un procédé.

le modèle de Workflow, finiront, éventuellement, par s'achever. Or, cette éventualité ne garanti pas leur terminaison. Ainsi, une instance spécifiant un procédé non défini dans le modèle de Workflow, qui n'est pas en mesure de s'achever, est une **exception**. Ceci pourrait être dû au fait que l'une des activités de l'instance se trouve dans l'un des cas suivant :

- ***Un échec système*** : Le rétablissement suite à une défaillance du système tels qu'une coupure de courant, une réinitialisation, un arrêt du serveur, *etc.*, est généralement traité au niveau de l'activité. Ce traitement repose sur la capacité de recouvrement des systèmes (bases de données) sous-jacents. Le Workflow peut se trouver temporairement suspendu, mais sa reprise est conditionnée par la récupération du système local ;
- ***Un conflit sémantique*** : Un conflit sémantique survient quand une instance est incapable de s'exécuter conformément à son modèle de Workflow. Ainsi, le modèle de Workflow n'est pas en mesure de répondre aux besoins particuliers de cette instance exceptionnelle.

Les changements du Workflow peuvent intervenir, alors, à la suite d'un conflit sémantique. Ce dernier peut être identifié par les utilisateurs du Workflow lors d'une exception, et/ou par des entités externes lorsque le processus (c'est-à-dire, le modèle) change, affectant l'ensemble ou certaines instances.

Pour mieux cerner le contexte du changement dans un Workflow, nous identifions relativement cinq types distincts de changements. Nous abordons ces mécanismes comme des politiques du changement ou de migration pouvant être adoptées par l'administrateur du Workflow (*Workflow Administrator* : WfA). Quand un procédé métier nécessite une modification, en raison de certains événements internes ou externes à l'organisation, les changements sont généralement planifiées, révisées, approuvées par les décideurs ou les analystes, et ce n'est seulement qu'ensuite qu'ils sont propagés au niveau opérationnel. Le rôle du WfA est vu en tant que médiateur entre les propositions stratégiques de la direction et la propagation de ces propositions au niveau opérationnel. C'est seulement alors, que le WfA doit être capable de traduire les changements du processus métier dans le modèle de Workflow, et de prendre les décisions adéquates au sujet des instances encore actives (c'est-à-dire, la classe d'instances concernées). Une **classe d'instances** est définie comme un ensemble d'instances qui peuvent être représentés par le

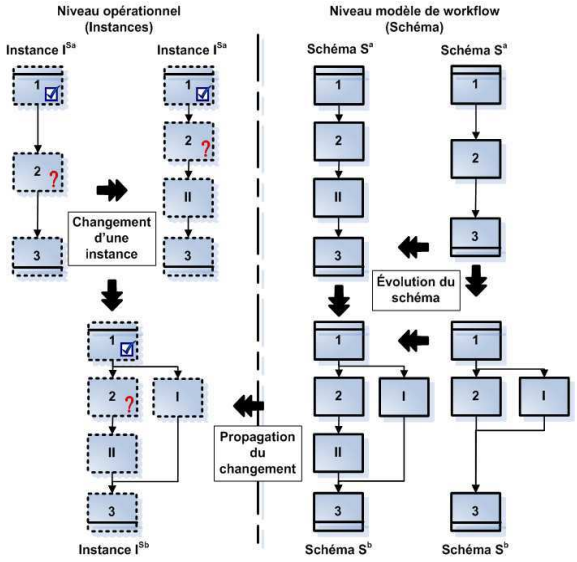


FIG. 1.3 – Le concept fondamental du changement.

même sous-graphe d'exécution. Les politiques du changement qui peuvent êtres adoptées par le WfA sont :

- **Mise à niveau** (*Flush*) : Dans ce cas de figure, on autorise un achèvement, selon le vieux modèle (c'est-à-dire, avant l'occurrence du changement), de toutes les instances en cours, mais les nouvelles instanciations sont faites conformément au nouveau modèle. Les nouvelles instances peuvent être, arbitrairement, mise en attente jusqu'à la terminaison de toutes les instances en cours. Toutefois, la cœxistence (cohabitation) des deux spécifications pourrait aussi être autorisée. Par exemple, les règlementations gouvernementales peuvent modifier la politique d'immigration en durcissant toute demande de régulation effectuée après une certaine date. Cependant, les demandes antérieures (c'est-à-dire, déposées avant la date en question) ne sont pas affectées par la nouvelle législation. Les demandes effectuées à compter de cette date seront conformes à la nouvelle loi, nécessitant ainsi, de disposer des deux schémas pendant la période de transition ;
- **Interruption** (*Abort*) : Lors de la modification du modèle, les instances actives du Workflow peuvent être interrompues. Généralement, cette politique est utilisée pour adapter une instance particulière, par exemple l'annulation d'une réservation. Cependant, elle peut également être le résultat d'un changement radical dans l'organisation, par exemple, en raison d'une précédente planification et de mauvaises procédures pratiquées, la gestion des achats pourrait nécessiter un changement. Pour surmonter une crise budgétaire, la nouvelle direction peut annuler tous les ordres d'achat en cours, de réaffecter le budget, et d'introduire une nouvelle procédure d'achat. L'annulation de la procédure d'achat causerait l'interruption des instances de Workflow des commandes actuelles, puis leur redémarrage selon la nouvelle procédure ;
- **Migration** (*Migrate*) : Ce cas de politique du changement affecte toutes les instances en exécution sans pour autant les interrompre. D'une manière évidente, lors de l'exécution du processus, les instances du procédé se trouvent à différents stades d'avancement. Un problème se pose quand une instance est à une étape où les tâches déjà réalisées ont affecté le procédé de façon à ce que les tâches ultérieures sont incapables d'agir conformément à la nouvelle spécification. Ainsi, afin de mettre l'instance en conformité avec la nouvelle spécification, il peut également s'agir de défaire ou de compenser les tâches accomplies ;
- **Adaptation** (*Adapt*) : Adapter inclut les cas d'erreurs et d'exceptions, où le modèle du processus ne peut pas changer de façon permanente, mais certaines instances doivent être distinctement traitées en raison de certaines circonstances exceptionnelles et imprévues. Par exemple, dans le

processus d'admission d'une université, il pourrait y avoir un candidat ayant une formation en systèmes d'information et en Informatique, et qui postule à une thèse en gestion. La commission d'évaluation du département des sciences de gestion peut renvoyer la demande, pour examen, à l'institut d'Informatique, afin d'évaluer le candidat. Ce genre de modifications ponctuelles (*ad-hoc*) dans les Workflows répétitifs et prévisibles est rarement tenu de se produire ;

– **Construction** (*Build*) : L'approche par construction d'un nouveau processus est également une classe des politiques du changement des procédés. La différence, par rapport aux autres approches, est que le point de départ n'est pas un modèle préexistant, mais une description élémentaire permettant de définir uniquement les bases, ou à la limite-même, un processus vide. Un exemple typique pourrait être un processus où les activités sont totalement identifiées, mais leur ordre d'exécution est le plus souvent inconnu. L'avantage d'une telle approche par construction, c'est qu'elle permet d'effectuer des changements sur des processus qui ne peuvent être entièrement prédéfinis. Essentiellement, le même mécanisme permet une définition dynamique (par construction), ainsi qu'une modification dynamique (migration, adaptation, *etc.*).

Le dénominateur commun entres toutes ces politiques, à l'exception peut-être de la mise à niveau, est qu'elles sont applicables aux instances actives d'un modèle de Workflow donné. Ainsi, elles dictent le champ d'application de l'évolution des Workflows [9] et représentent des modifications dynamiques ou à la volée (*On-The-Fly*). Contrairement aux modifications statiques, qui sont simplement un changement au niveau du modèle de Workflow. Autrement dit, aucune considération n'est faite pour les instances actives. C'est donc à travers ses deux facettes qu'est abordé le problème de l'évolution du Workflow :

– **L'évolution statique du Workflow** revoit à la problématique de la modification de la description du modèle de Workflow. Le WfMS doit alors fournir un langage permettant le raffinement progressif du procédé, sans pour autant nécessiter sa réécriture à chaque stade de l'évolution. Il doit cependant garantir que les versions du modèle sont syntaxiquement correctes ;
– **L'évolution dynamique du Workflow**, quant à elle contribue à la gestion des instances dont la description est susceptible de changer. Dès lors, le WfMS devrait assister les designers, par des mécanismes appropriés, à pouvoir « soigneusement » modifier les instances en cours d'exécution pour quelles répondent aux nouveaux besoins de la modification.

1.1.3 Le Workflow dynamique

Dans ce qui suit, nous allons examiner certaines des questions critiques de la gestion des Workflows dans un environnement dynamique.

Le changement et le temps sont les deux principaux éléments qui rendent les Workflows dynamiques. En d'autres termes, des changements incontrôlés sur modèle de Workflow ou sur les instances peuvent conduire à des situations inconsistantes ou erronées. Le problème est plus grave encore, lorsque les instances en cours d'exécution y sont impliquées.

1.1.4 Un langage pour le changement

Un langage adapté pour le changement est fondamental pour le support de la dynamique des modifications d'un Workflow. Ce langage doit être composé d'un **ensemble complet et minimal d'opérations**, capables d'exprimer tous les changements autorisés, appelées primitives de changement. Pour ce faire, deux enjeux sont à considérer. Premièrement, identifier un jeu complet d'opérations minimales, et comment les concevoir. Cet ensemble de primitives doit être : complet, minimal, et cohérent. La **complétude** exprime la possibilité de transformer un schéma générique du Workflow en un autre schéma générique. La **minimalité** représente l'accomplissement de la complétude avec un ensemble réduit de primitives. Alors que, la **cohérence** aborde le problème d'effectuer des modifications sur le modèle de Workflow sans causer des erreurs lors de sa compilation (*Compile-Time Errors*) ou de son exécution (*Run-Time*

Errors), comme par exemple, des interblocages ou des invocations incorrectes de ressources. Par emprunt aux concepts de l'évolution du schéma dans les bases de données objets (se reporter à l'annexe D), pour ne citer que ceux-là, il est intéressent de faire la différence entre les notions de : cohérence structurelle et de cohérence comportementale [9, 23] d'un côté, et de cohérence sémantique [24], de l'autre.

La cohérence structurelle, comme un concept appartenant à l'aspect statique du Workflow (c'est-à-dire, au modèle de Workflow), est la propriété, qu'après chaque séquence de modifications, le schéma en résultat est légal. Un schéma est considéré comme légal lorsque toutes les activités du modèle de Workflow sont accessibles. Cette notion d'accessibilité implique l'existence d'un chemin entre l'activité initiale et vers toutes les autres activités du schéma.

La cohérence comportementale est une notion appartenant à l'aspect dynamique du Workflow (c'est-à-dire, la population des instances en cours d'exécution). Elle implique, que peu importe l'ensemble de primitives, l'application de ces primitives sur une instance en exécution produit une instance (en exécution) conforme à un schéma différent dont l'état est considéré autorisé (dans le sens de sa légalité). L'état d'une instance I d'un schéma S est légal si elle peut évoluer sans provoquer d'erreurs d'exécution. Les cohérences ; comportementale et structurelles doivent être garanties lors de l'application de chaque primitive, prise individuellement. Ce qui permet de généraliser la validité de ces propriétés sur une séquence de primitives.

Pour comprendre le concept de la **cohérence sémantique**, prenons l'exemple, présenté par la figure 1.4, d'une instance I d'un schéma S modélisant le traitement médical d'un patient donné. Supposons, qu'en raison de l'apparition soudaine, chez le patient, de maux de tête, l'infirmière décide de lui administrer de l'*Aspirine*. Elle insère, par exemple à travers son poste de travail, une activité « Administrer de l'*Aspirine* » dans l'instance I d'une manière *ad-hoc*.

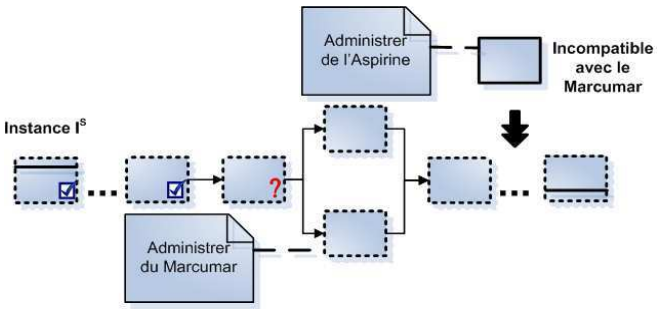


FIG. 1.4 – Exemple de changements dans un cas de procédé pour le suivi médical.

Toutefois, dans ce procédé de traitement, le médicament *Marcumar* incompatible avec l'*Aspirine*, à été déjà administré dans les activités antérieures : il y a là un **conflit sémantique**. Même si le changement du procédé est syntaxiquement correct, il ne l'est pas sémantiquement. Ceci se produit souvent, surtout quand l'instance est modifiée en fonction des besoins inopinés (par exemple, l'administration du *Marcumar*) ou lorsque des changements du procédé se produisent simultanément (ce que l'on a désigné par changements concurrents dans la section 2.3) au niveau du schéma et au niveau d'une instance. Ainsi, il est tout à fait probable que ce genre de conflits ne soit décelable par les utilisateurs. Il est donc nécessaire que le système de gestion de procédé soit conscient de ce type d'incompatibilités entre les activités. [25] proposent qu'il y est deux méthodes pour assurer une correction sémantique sur le schéma du procédé selon les deux types de construction du modèle de Workflow. Si un modèle de procédé est construit par application des changements à un schéma « vide », le PMS devrait effectuer une vérification sémantique à chaque occurrence d'une opération de changement et vérifier si la correction sémantique sur le schéma est préservée. La deuxième possibilité est de construire le modèle à partir d'un modèle existant. Ceci est possible par l'importation d'un modèle existant dans le PMS ou en utilisant des techniques de fouille de données sur les exécutions [26] pour en extraire des connaissances sur l'histoire de l'évolution des procédés et s'en servir pour optimiser le design de ces derniers (*Process Mining Techniques*). Il s'agit

alors de vérifier, en une seule fois, la correction de la totalité du schéma du procédé. Cette vérification de la correction se fait par un contrôle des contraintes sémantiques définies en amont sur le processus.

1.1.5 Le problème du changement dynamique

A un niveau conceptuel, si *W. M. P. van der Aalst* pose le problème majeur du changement dynamique (*Dynamic Change Problem*) [13], qui consiste en la difficulté à adapter les instances du procédé en exécution au nouveau schéma (c'est-à-dire, comment transférer le modèle de ces instances vers une nouvelle version du modèle de procédé), au niveau opérationnel, bien d'autres défis ont étés pointés par *S. Rinderle et al.* [12], dont les principaux sont :

- **Remise en cause du passé** (*Changing The Past*) : Ce problème correspond à la règle dictant le fait que l'exécution antérieure (avant l'occurrence du changement, ou de l'exception) d'une instance ne peut être remise en cause ou défaite. Négliger cette règle pourrait provoquer des états incohérents de l'instance (par exemple, des exclusions mutuelles ou des interblocages) ou des données manquantes dans les conteneurs d'entrées des activités invoquées (voir figure 1.5.a) ;
- **Des états de flottement** (*Dangling States*) : Ce problème se pose dans les métamodèles de Workflow ne faisant pas la distinction entre les états « activée » et « démarrée » d'une activité. En conséquence, et très souvent, ces approches vont soit interdire la suppression des activités activées, ce qui est très restrictif, soit permettre la suppression d'activités déjà démarrées, conduisant ainsi à une perte du travail réalisé entre temps (voir figure 1.5.b) ;
- **Insertion en parallèle** (*Parallel Insertion*) : Par opposition au changement de l'ordre, ce problème se pose lors de l'insertion d'une nouvelle branche parallèle dans le flot existant. Par exemple, dans une modélisation utilisant les réseaux de Petri, et après un tel changement, il se peut qu'il y soit une nécessité à ajouter des jetons supplémentaires pour éviter des interblocages dans la suite de l'exécution (voir figure 1.5.c) ;
- **Tolérance aux boucles** (*Loop Tolerance*) : Ce problème fait référence au potentiel d'une approche à pouvoir gérer des changements se produisant dans des boucles du modèle de Workflow. En particulier, les approches ne doivent pas arbitrairement exclure certaines instances de migrer vers un nouveau schéma sous prétexte que les modifications concernent des boucles dans le flot de contrôle (voir figure 1.5.d) ;
- **Changement de l'ordre** (*Order Changing*) : Ce problème renvoie à la façon d'adapter correctement le marquage d'une instance lors du changement de l'ordre des activités comme la mise en parallèle, en série, ou la permutation des activités (voir figure 1.5.e).

Les deux problèmes du changement de l'ordre et l'insertion en parallèle sont étroitement liés à ce qui est désigné dans [27] comme « **le bogue du changement dynamique** » (*Dynamic Change Bug*).

1.1.6 Comment effectuer le changement ?

Avant de pouvoir effectuer un changement il nécessaire, de définir la modification à opérer sur le procédé. En précisant la politique du changement, en spécifiant les instances affectées dans le cas d'une interruption ou une adaptation et en déterminant les changements à apporter au modèle de Workflow, il existe deux contraintes évidentes à respecter : les modifications à apporter doivent être connues, et les changements doivent être vérifiés conformément à une propriété de correction du langage de modélisation utilisé.

A cet égard, un bon nombre de recherches ont été faites sur le plan conceptuel de la spécification des modèles de Workflow et ont fournis un ensemble d'opérations de changement garantissant une correction (structurelle) du modèle du changement.

Généralement, toute modification structurelle sur un procédé peut être exprimée, par exemple, selon les opérations élémentaires suivantes, constituant ainsi un ensemble minimal et complet de primitives :

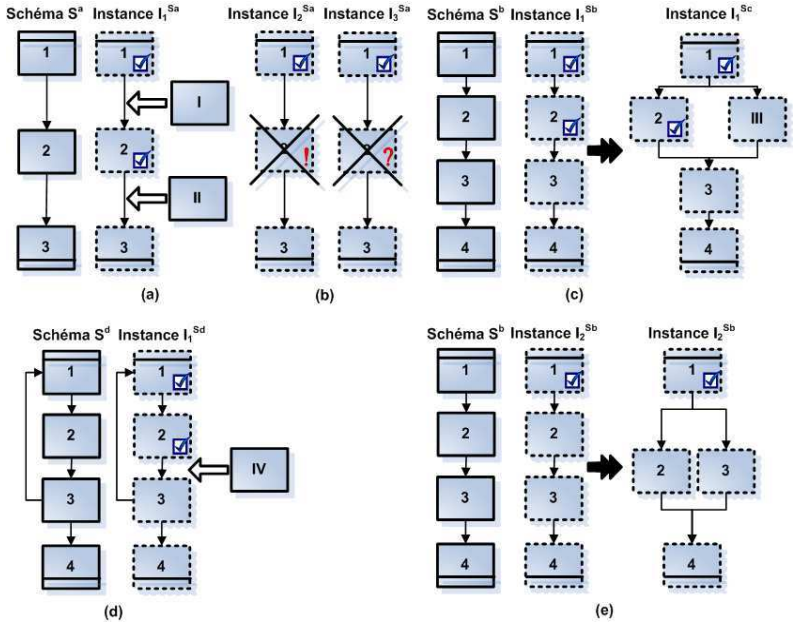


FIG. 1.5 – Les principaux problèmes du changement dynamique.

- Insertion d'une nouvelle activité : *AjouterActivité (Activité t)* ;
- Suppression d'une tâche existante : *SupprimerActivité (Activité t)*, décision incluse sur le fait, si une suppression automatique doit être réalisée sur une branche du flot, lorsqu'il n'existe plus de chemin partant de l'activité initiale vers une des activités appartenant à cette branche ;
- Modification des propriétés de la tâche (données nécessaires, application sous-jacente, d'affectation des ressources) : *AjouterVar (NomVar v, Valdéfaut d)*, *SupprimerVar (NomVar v, Valdéfaut d)* ;
- Modification de l'ordre d'exécution des tâches : *AjouterSuccesseur (Activité t, Activité s)*, *SupprimerSucesseur(Activité t, Activité s)*, *ModifierType(Activité t, type T)*, *ModifierCondition(Activité t, Condition C)*.

Les trois premières constituent une famille de **primitives déclaratives**. Alors que la quatrième, est la famille de celles qui sont dites « **des primitives de flot** » [9] influant sur la structure du schéma du Workflow. Le traitement de ces primitives est réduit à des manipulations de nœuds et de transitions. Dans la figure 1.6, nous présentons quelques exemples de ces opérations de changement, dans le seul

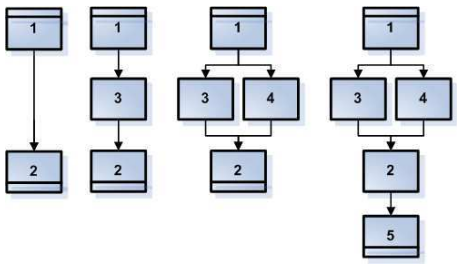


FIG. 1.6 – Exemples de changements : insertion d'activités dans le flot de contrôle.

but d'illustrer la diversité d'expression de ces opérations. Par exemple, l'ajout d'une tâche au procédé (figure 1.6, de gauche à droite) pourrait être aussi simple que l'insertion d'une nouvelle activité au modèle

(l'activité 3 dans la figure 1.6),en série, entre deux activités existantes (les activités 1 et 2 dans la figure 1.6). Ou bien, elle pourrait constituer l'ajout d'une activité concurrente (l'activité 4 dans la figure 1.6), et donc, implicitement, ajouter une nouvelle branche de flot. De même, l'ajout d'une nouvelle activité peut amener, seulement sous certaines conditions données, à y adjoindre un nœud de condition. Par analogie, la suppression d'une activité pourrait être considérée de droite à gauche sur la figure 1.6. Lors de la modification de l'ordre d'exécution (voir figure E.2 de l'annexe E), les activités séquentielles peuvent êtres mises en parallèle, ou les conditions sur les données peuvent êtres changées. Ainsi, dans n'importe quel langage de modélisation, toutes ces opérations vont effectivement constituer un jeu d'opérations primitives (ajouter un nœud, supprimer un nœud, ajouter une branche au flot, supprimer une branche du flot). Le choix de l'ensemble et de l'ordre des primitives de changements à appliquer pour une même opération de modification sera dicté par les propriétés sémantiques et le critère de correction du langage utilisé. Pour une description détaillée et une formalisation du changement, veuillez vous reporter à l'annexe E.

1.2 Problématique

Comme expliqué plus haut, du fait de l'hétérogénéité des étapes qu'implique un procédé métier et de la demande constante pour l'optimisation des processus, la flexibilité et l'adaptabilité deviennent les besoins majeurs aussi bien de tout système qui se veut gérer des Workflows (WfMS) ou être conscient du procédé métier (PAIS), que de son langage de modélisation du Workflow sous-jacent. Généralement, la flexibilité dans un WfMS implique deux aspects fondamentaux :

- La spécification d'un comportement d'exécution flexible permettant d'exprimer, dès la modélisation, une conduite précise et peu restrictive, mais cependant, correcte. Par exemple, pouvoir supporter une coopération entre activités d'un environnement centré sur le processus ;
- L'évolution des modèles du procédé pour permettre de modifier, d'une manière flexible, la spécification du Workflow aussi bien au niveau du schéma, qu'au niveau des instances pour répondre aux besoins de la réingénierie des activités ou aux situations de changements dynamiques du processus réel.

Dans le cadre de ce mémoire, on s'intéressera particulièrement à l'évolution des modèles des Workflows, sans pour autant occulter le premier aspect, car comme nous le présenterons plus loin l'approche que nous proposons est générique par rapport aux langages de modélisation des Workflows. Faire évoluer les modèles du Workflow est nécessaire pour plusieurs raisons. La plus importante est, sans doute, l'obligation de répondre au besoin de l'amélioration constante du processus métier, qui implique une adaptation constante des spécifications des Workflows correspondants. Une autre raison, pas moins importante, est le fait que le caractère prévisible du modèle du procédé (autrement dit, la rigidité imposée par une planification rigoureuse des activités à réaliser) requiert souvent un raffinement ou un ajustement de la structure conditionné par une situation particulière (*ad-hoc*) de l'environnement opérationnel. Bien d'autres raisons subsistent, comme la nécessité de corriger des erreurs de spécifications dans le modèle du Workflow et de paramétrer ou adapter le Workflow aux besoins d'un cas d'exécution particulier.

Par conséquent, l'évolution du workflow conduit à deux difficultés majeures : La première, est la gestion des différentes versions du schéma du Workflow suite à des changements, et des mécanismes (ce que nous avons désigné par politiques du changement dans la section 1.1) de propagation des ces mêmes changements aux instances. La seconde problématique est d'assurer que les modifications sur le Workflow se font d'une manière dynamique, à savoir que le changement doit être appliqué sur des instances en exécution.

1.3 Hypothèses

Pour être plus pertinent, nous restreignons le cadre de la solution que nous proposons afin de résoudre les problèmes de l'évolution des Workflows (énoncés plus haut) aux aspects du changement relatif au flot de contrôle. Bien que nous nous imposons cette restriction, l'énoncé de notre travail, comme nous le présenterons plus loin, pourra facilement être enrichi par des mécanismes pouvant offrir et supporter des changements au niveau des autres composants d'un Workflow (à savoir, le flot de données, les conditions, les rôles, les agents, les ressources, *etc.*. Se reporter à la section 1.1.1). Les approches existantes, que nous présenterons dans le chapitre suivant, abordent des problèmes distincts du changement dynamique selon des points de vue différents. D'une part, certaines s'intéressent à la cohérence de la propagation de l'évolution du schéma aux instances, et d'autres, s'intéressent à la cohérence du changement se produisant au niveaux des instances en cours d'exécution (dites biaisées, voir annexe E pour plus de détails). À la différence de ces approches, et pour aller outre ces spécificités, nous faisons l'hypothèse conjointe que :

- L'évolution du schéma du procédé doit être propagée dynamiquement aux instances en exécution, ainsi que ;
- Les instances peuvent évoluer individuellement et être concernées par des changements *ad-hoc*.

Il doit alors être possible qu'une instance puisse assimiler les changements du schéma, et que ses changements individuels puissent être propagés au schéma. Il faut s'avoir qu'on ne s'intéresse ici, qu'à la cohérence structurelle et comportementale du changement (voir section 1.1.4). Une omission volontaire est faite, et que nous reportons à des travaux ultérieures, de la problématique liée à la cohérence sémantique du changement dans les Workflows.

1.4 Conclusion

Considérer ce point de vue, nous vaudra la possibilité de construire un *framework* pour le changement, et pouvoir représenter l'évolution d'un modèle de Workflow par un système d'opérations génériques. Cette généricité s'exprime, premièrement, par une abstraction vis-à-vis d'un langage particulier de modification du Workflow (c'est-à-dire, d'un ensemble complet et minimal de primitives. Lire section 1.1.4). Deuxièmement, une généricité par rapport au métamodèle du Workflow (c'est-à-dire, à tout composant ou langage de modélisation), car ce sont les algorithmes des opérations qui capture la sémantique inhérente au formalisme employé par un WfMS (voir section 1.1.1) et non plus le modèle du *framework* du changement lui-même. Troisièmement, une généricité par rapport à toute politique du changement (voir section 1.1), car par extension, le WfA pourra puiser dans un catalogue d'opérations prédéfinies lui permettant d'adapter le modèle du *framework* à sa convenance. Bien évidemment, il nous est nécessaire de définir de nouvelles notions de conformité et de corrections reposant sur la notion d'opérations.

Dans le chapitre suivant, que nous consacrerons à l'étude du contexte de la flexibilité des Workflows, nous allons motiver notre proposition pour un modèle innovant, qui permet d'élever les paradigmes existants à un niveau d'abstraction nécessaire à une gestion encore plus flexible du changement métier. Ainsi, nous présenterons notre *framework* générique basé sur les opérations qui permet de garantir une flexibilité des WfMS pour le cas particulier des changements structurels.

Chapitre 2

État de l'art

Sommaire

Dans ce chapitre nous allons présenter les enjeux de la dynamique des procédés métiers. À travers l'état de l'art des approches existantes, nous montrerons comment la gestion des changements influence la flexibilité des modèles de Workflows et comment l'adaptabilité des procédés métiers est résolue dans la littérature. D'abord, nous orientons nos propos vers une typologie des procédés métiers pour mettre en évidence le besoin d'adaptabilité des systèmes de gestion de Workflows. Ensuite, nous nous intéresserons à la gestion du changement comme une solution pour la flexibilité des Workflows. Enfin, nous dégagerons la notion d'adaptabilité des Workflows, et nous motiverons la nécessité de recourir à des métamodèles permettant un contrôle accru de leurs exécutions flexibles.

2.1 Typologie des Workflows

Plusieurs taxonomies des Workflows subsistent, particulièrement D. *Georgakopoulos et al.* [28] les classifient, d'une manière générale, en trois principales catégories : « ***ad-hoc*** », **procédural** et **administratif**, selon les trois dimensions de répétitivité, de prévisibilité (ou de prédictibilité) et de fonctionnalité. Cette classification oppose les Workflows comme étant orientés système à ceux qui sont orientés utilisateur [6]. Cependant, on pourrait rajouter une catégorie supplémentaire de Workflows qualifiée de **collaboratif** [29, 30]. Bien que la dimension collaborative permette d'évaluer le degré de criticité et de complexité des Workflows, elle ne peut exprimer, toutefois, une propriété cruciale des procédés métiers : la rigidité. Le niveau de rigidité dicte ainsi la nature d'un procédé.

Dans le cas d'un Workflow « procédural» (aussi appelé Workflow de production ou Workflow directif) correspondant à des processus métiers connus de l'entreprise et faisant l'objet de procédures pré-établies. Le cheminement du Workflow est, dans ce cas, plus ou moins figé, et est totalement connus avant même son exécution. D'autre part, pour un Workflow « *ad-hoc* », ou stratégique, ou même un processus essentiel à la mission (*Mission Critical Process*) qui se base sur un modèle collaboratif dans lequel les acteurs interviennent dans la décision du cheminement, le cheminement du Workflow est dynamique. Il ne peut en aucun cas être prédis ou fixé à priori. Sinon, et dans le cas des procédés dits « hybrides » qui peuvent être décrits par l'expression : « c'est de la manière suivante que nous faisons les choses ici ! » [11]. Ceux-là étant définis en amont de l'exécution, permettent cependant une certaine flexibilité [6]. Ce genre de procédés (par exemple, la réservation de voyage, le traitement des demandes de règlement d'assurances

ou l'admission des étudiants dans une université) est souvent un bon candidat à l'automatisation en utilisant la technologie des Workflow. Malgré cela, et à la vue des avancées technologiques effrénées, l'inconstance des besoins et des réglementations, et l'introduction de nouvelles méthodes de travail, les procédés métiers sont constamment révisés, optimisés et adaptés à leur environnement instable. De ce fait, la stabilité-même de ces procédés définis comme rigides, répétitifs ou prévisibles se voit, et doit être remise en cause.

D'un autre point de vue, les organisations visant à adopter une technologie de Workflow doivent, inconditionnellement, recourir à des investissements significatifs en terme de temps, d'effort et de ressources pour réussir son déploiement. C'est alors qu'il devient essentiel de s'interroger sur une constatation. Dans un environnement aussi dynamique de part sa nature, la technologie employée permet-elle une supervision adéquate à laquelle aspirent tous les décideurs, ou engendre-t-elle une structure du procédé si rigide, qu'il devient nécessaire d'allouer des ressources supplémentaires pour pouvoir maintenir des outils de support additionnels ? Et quel-est le degré de pertinence d'un processus, si dès lors qu'il est mis en place, il est susceptible d'être changé ?

Semblablement, se pose une troisième question : comment saisir la dynamique du procédé métier au sein du modèle de Workflow ? Une première approche possible serait d'admettre une certaine **flexibilité dans la définition du Workflow**, qui permettra d'éviter au plutôt les potentielles modifications ultérieures. Cette approche est peu satisfaisante, car il n'est plus possible de faire la distinction entre le workflow *ad-hoc* et le workflow procédural. Accepter ce niveau de flexibilité, lors de la modélisation, engendrerait, au moment de l'exécution, l'impossibilité de garantir un niveau de contrôle ou de coordination suffisant sur les activités. Plus encore, autoriser un Workflow à fonctionner librement, dépourvu de tout contrôle ou de toutes contraintes, mènerait à l'anarchie et remettrait en cause la raison-même pour laquelle la technologie des Workflow eut été introduite.

Inversement, des modèles hautement normatifs imposeraient un contrôle trop rigide, même-s'il s'agit de modéliser des flux de travail répétitif. Ce genre de modèles inflexibles, est contre-productif pour la réactivité et la capacité d'adaptation, qui sont vitales à chaque organisation et qui lui permettent de faire face aux particularités de son environnement concurrentiel. La question est alors de savoir à quelle degré de granularité le modèle de Workflow doit-il se positionner ? Trouver un compromis entre une définition précise et une définition flexible est un problème difficile à résoudre pour lequel les solutions génériques ne peuvent être adéquates.

Une autre approche est d'**anticiper les changements** et/ou les exceptions pouvant survenir, et de les intégrer dans la logique du procédé. Toutefois, les exceptions sont souvent *asynchrones* par rapport au flot de contrôle, les prendre en compte dans le modèle de Workflow serait irréaliste vu leur grand nombre. Toutefois, même s'il était possible de définir toutes les exceptions au sein du procédé, cela provoquerait une explosion combinatoire du modèle à construire. Et même encore, parfois, il serait nécessaire d'inclure certains écarts de circonstance au modèle initialement conçu.

Étant donné qu'il doit être conscient de son environnement, tout modèle de Workflow tente, du moins, de saisir les aspects fonctionnels, temporels et organisationnels des procédés métiers régissant les organisations pour lesquelles il est destiné. Ainsi une modification survenant dans le procédé peut, en conséquence, avoir un impact sur l'un de ces aspects. Une modification peut concerner un changement au niveau des applications sous-jacentes, les rôles et les responsabilités des utilisateurs, le flot de données des activités, *etc.*. Néanmoins, nous visons ici les changements apportés aux aspects fonctionnels et/ou comportementaux du procédé métier, c'est-à-dire des modifications sur la structure du Workflow. Ceux-ci sont les plus difficiles, aussi bien à définir qu'à mettre en œuvre.

Le facteur temps [31] n'est pas à négliger, et est peut-être le deuxième facteur critique dans la plupart des systèmes de gestions de procédés métier. La gestion efficace du temps dans les Workflow demeure un défi, puisque souvent, la prise de décision contrainte par le temps est en elle-même un processus dynamique et complexe. Notamment, lorsque des changements sur la structure du procédé se produisent, la gestion du temps peut impliquer de tirer des conclusions à la volée sur la base d'informations incomplètes. La spécification et l'exécution sous garantis des contraintes temporelles deviennent alors primordiales pour le succès du déploiement des Workflows. Cela ajoute, alors, une dimension supplémentaire de complexité

au problème du changement dans les procédés métier.

Devant la diversité des problématiques que pose le changement dans les procédés métiers, et pour être le plus pertinent possible, dans ce mémoire, nous abordons essentiellement les questions et les problèmes liés à la gestion des Workflows dynamiques. Nous étudierons les changements et leurs impacts sur les contraintes structurelles. Dans les sections qui suivent, nous traiterons du :

- contexte du changement dans les Workflows, qui est présenté comme une taxonomie des modifications se produisant au niveau métier et leurs effets sur les WfMS sous-jacents ;
- la modélisation des Workflows dans une situation évolutive, de l'impact du choix d'un métamodèle adéquat, et de l'incertitude sur le processus du changement ;
- la faisabilité d'un support automatique à la propagation des changements aux implémentations sous-jacentes.

2.2 Gestion du changement dans les Workflows

Si dès 1995, *C. Ellis et K. Keddara* [6] ont pointé le problème du changement dynamique dans les systèmes de gestion de procédés (voir section 1.1.5), et ont posé les premiers jalons de sa formalisation. Toutefois, leur tentative ouvrait le champ à des « recherches sur les différentes notions de correction, et l'existence d'un ensemble complet de modifications élémentaires qui peuvent, sous certaines conditions, garantir la correction tout en étant assez puissants pour modéliser des changements complexes». Les travaux qui suivront aborderont plus en détails la thématique de la flexibilité dans la gestion des procédés.

Le projet *ADEPT*, à l'Université de *Ulm* vise à offrir un support de **Workflow flexible** pour les applications avancées, principalement dans le contexte des environnements hospitaliers. Le modèle de Workflow est spécifié en langage *ADEPT Workflow*, et les modifications dynamiques du procédé sont modélisées en utilisant $ADEPT_{\text{flex}}$ [3]. Ce formalisme se compose d'un ensemble de règles, de critères et de méthodes permettant d'effectuer des modifications dynamiques sur les instances de Workflow. Une série d'opérations de changements, basées sur le langage *ADEPT Workflow*, est spécifiée dans le *framework* $ADEPT_{\text{flex}}$. Selon ce modèle logique, les opérations de **changements structurels sur les instances en exécution** (*Instance-Specific Changes*) peuvent être réalisées, d'une manière contrôlée, par les utilisateurs. Par exemple, l'optimisation d'un Workflow avec une tâche implique d'incruster une nouvelle activité dans le modèle. L'insertion de cette activité dans le Workflow se fait par la définition d'un ensemble d'activités qui doivent être accomplies avant le début de la nouvelle activité, et d'un ensemble d'activités qui ne peuvent démarrer qu'à la terminaison de la nouvelle. Le travail conceptuel est validé par l'implémentation d'un prototype. Outre le fait que *ADEPT* [32] ne prévoit pas de **modification du schéma** de Workflow (*Schema Changes*), la portée de chaque modification n'est défini que sur une seule instance. Une approche présentée dans [4, 33, 34, 35] est plus générale que *ADEPT*, car aucune structure symétrique n'est imposée sur le schéma du Workflow. Celle-ci prétend améliorer la flexibilité des systèmes de gestion de procédés en offrant la possibilité de changer dynamiquement aussi bien la structure des instances que le schéma. Ainsi, à différents instants, il est possible de contrôler une instance par différents schémas. Toutefois, à un instant précis, une instance n'est rattachée qu'à un unique schéma. Cette caractérisation permet de maintenir une propriété de correction des instances par rapport au schéma, tout en permettant l'**adaptation dynamique** (se reporter à la section suivante).

L'étude de [9], a pu aborder sous un angle conceptuel (c'est-à-dire, sans la validation par un prototype opérationnel) la flexibilité, en général, dans la gestion des Workflows, et l'évolution du Workflow, en particulier. Dans cette approche, la flexibilité est fournie par l'évolution du Workflow. L'idée de cette évolution est similaire à l'adaptation dynamique [33] qui étant donnée une instance et son schéma, comment adapter (ou migrer), cette instance vers un autre schéma modifié. Toutefois, l'approche est moins générale que celle de l'adaptation dynamique. Bien que l'approche présentée dans [9] utilise des opérations spécifiques de modification (appelées, **primitives d'évolution du Workflow** : *Workflow Evolution Primitives*) qui sont appliquées au schéma initial afin de le modifier (voir section 1.1.4). Toutefois, elle diffère de l'adaptation dynamique, dans le sens où cette dernière permet d'ajuster une instance à un schéma arbitraire, sous condition d'un **critère de cohérence** [4]. Ce critère de cohérence (*Consistency Criteria*)

se fonde sur une évaluation entre l'instance et le nouveau schéma du Workflow. De plus, il existe une autre **propriété de conformité restrivtive** (*Restrictive Compliance Property*), introduite dans [9], qui permet de détecter les instances devant êtres **migrées** vers le nouveau schéma, obtenu après l'application des opérations de modification.

Une autre approche intéressante en vue des modifications dynamiques est présentée dans [30]. Elle propose un mécanisme fondé sur des règles, qui à une **exception sémantique** donnée (voir section 1.1.4), permet de calculer l'ensemble des instances et les modifications nécessaires à apporter sur cet ensemble en vue de faire face à cette situation exceptionnelle. Toutefois, les exceptions et les règles de modification doivent être connues au moment de la modélisation. Lors de l'occurrence d'une exception sémantique, un système fondé sur ces règles décide si certaines activités peuvent être abandonnées ou bien lesquelles doivent être démarrées. Dans un deuxième temps, l'ensemble des instances concernées par cette exception est déterminé. Dans une troisième étape, pour chaque instance affectée, le où les changements devant être effectuer sont déterminés. Enfin, les modifications sont appliquées aux instances afin de les adapter à la situation exceptionnelle. C'est alors qu'elles peuvent poursuivre leurs exécutions.

M. de Leoni et al. [36] ouvre les perspectives vers une approche générale, fondée sur le **suivi de l'exécution** (*Execution Monitoring*), **pour l'adaptation automatique** des processus dans les scénarios dynamiques. Contrairement à la plupart des approches précédentes, celle-ci ne requiert pas la définition de la stratégie d'adaptation dans le processus lui-même (lors de la conception) pour planifier les changements probables (lors de l'exécution). Elle présente, assez exhaustivement, comme dans [12] d'ailleurs, une comparaison des principales approches pour le support du changement dans les systèmes de gestion de procédés, les classifiant ainsi selon leur potentiel d'adaptation. Ainsi, l'**adaptabilité** (voir section 1.1.2) dans les PMSs peut être considérée à deux niveaux : au niveau du schéma du Workflow et de celui des instances. Les changements au niveau du schéma sont, par exemple, nécessaires pour adapter le PMS à l'optimisation des procédés métier ou à de nouvelles lois [37]. En particulier, les applications pour le support des processus de longue durée (par exemple, la gestion des emprunts bancaires ou de traitements médicaux) [32] et les instances qu'elles contrôlent sont affectées par ce type de changements. Par opposition, les changements sur une seule instance du processus (par exemple, pour insérer, supprimer ou déplacer une étape particulière du processus) sont souvent appliqués en fonction de circonstances singulières afin de faire face à une situation exceptionnelle (par exemple, la déconnexion des pairs dans les réseaux mobiles, l'évolution des besoins du processus).

2.3 Adaptation des Workflows

Les auteurs de [11, 22] affirment « qu'un métamodèle soigneusement choisi» pourrait réduire considérablement la charge du travail ultérieure lors de l'occurrence des modifications. D'autre part, une omission ou une faute compromettrait l'apport de la technologie de Workflow. Cela pourrait aboutir à des problèmes encore plus graves, lorsque les améliorations de la conception ou des changements de procédés ne peuvent être intégrés au modèle en raison des limites du métamodèle.

Pour offrir un support à l'adaptabilité des Workflows les approches actuelles se basent sur différents métamodèles. Très souvent, les solutions offertes par les PMSs sont tributaires aussi bien de l'expressivité, que de la sémantique du formalisme employé. Dans [12], une taxonomie classifie ces approches en deux classes, selon leur sémantique opérationnelle et leur stratégie d'exécution. Il est à noter que, dans un moteur d'exécution, le déroulement effectif des instances est matérialisé par des déclenchements successifs des activités. Au niveau opérationnel, ces activations sont concrétisées par un relais (sous contraintes des conditions sur le flot de contrôle et le flot de données) de certains types de structures appelés : jetons.

Dans ce sens, la première classe, celle des **jetons-vrai** (*True-Tokens*), est une stratégie utilisant un seul type de jetons (de flot de contrôle) pour chaque instance du Workflow. En revanche, la seconde stratégie est fondée sur deux types de jetons : les jetons-vrai et les **jetons-faux** (*False-Tokens*). Explicitement, les jetons-vrai déclenchent les activités devant êtres franchises, alors que les *jetons-faux* décrivent celles qui doivent êtres omises ou abandonnées.

Les modèles qui utilisent seulement la **sémantique-vrai** (c'est-à-dire, les *jetons-vrai*) incluent, par exemple, le formalisme des réseaux de Petri (*Petri-Nets*) plus ou moins étendus. Parmi ces approches, on peut trouvé :

- Les ***WF Nets*** [7, 13, 27] admettent qu'un schéma est constitué d'une seule activité initiale (marquant le début du procédé) et d'une seule activité finale (exprimant la terminaison du procédé). Ainsi, pour garantir la terminaison du *WF Net*, le réseau doit d'être conforme à une propriété appelée (*Soundness Property*) [7] exprimant le fait que, selon un marquage donné, toutes les places (par conséquence, les activités du procédé) sont toujours atteignables et qu'il ne contient pas de verrous. C'est-à-dire, qu'à la terminaison les jetons ne sont contenus que dans une seule place ;
- Les ***Flow Nets*** [6, 38, 39] : Leur sémantique opérationnelle est comparable aux *WF Nets* mais avec une différence majeure : Il peut y avoir plus d'un jeton dans une place. *Chautauqua* [39] propose une implémentation, où les *Flow Nets* sont généralisés à des *Information Control Networks* ;
- Les ***MILANO Nets*** [8] : une autre approche basée sur une extension des réseau de Petri colorés.

Un fait intéressant, est que ces approches font abstraction des états internes d'une activité. Autrement, elles ne font la distinction qu'entre des transitions activées ou non activées. Les approches qui, en plus, utilisent les *jetons-faux* pour représenter le saut d'activités ou de branches sont empruntées aux métamodèles de graphes ou de réseau d'activités. Par opposition à la *sémantique-vrai,* ces métamodèles font la distinction entre différents **états du cycle de vie d'une activité**. Généralement, l'activité est définie comme **inactive** (*NotActivated*) dans son état initial. Elle est « **activée**» (*Activated*) lorsque toutes ses préconditions sont vérifiées. Quand cette dernière démarre son état « **executée**» (*Running*). Enfin, lorsque les postconditions sont satisfaites, l'état est alors « **terminée**» (*Completed*) (voir figure 1.2). En outre, pour représenter l'omission d'une activité, les modèles imposent un état « **abandonnée**» (*Skipped*). En plus, un **historique d'exécution des activités** est maintenu pour chaque instance. En fonction de la manière dont-elle représente les jetons, cette stratégie de la **sémantique-vrai/faux** se subdivise en deux sous-classes d'approches. Une première possibilité, **basé sur l'historique** (*Based on History Logs*), est l'utilisation de journaux d'événements sur le début et l'achèvement des activités, ce qui permet d'obtenir une construction de jetons à partir de l'histoire d'exécution : ***WIDE Graphs*** [9], ***TRAMs Graphs*** [10]. Sinon, l'autre alternative est l'emploi d'un **marquage spécifique** des activités (c'est-à-dire, inhérent au modèle employé : *Model-Inherent Markings*) pour les besoins de la journalisation [3, 11, 4], comme dans les ***WASA$_2$ Activity Nets*** [4, 33, 34], les ***Breeze Activity Nets*** [11, 22], ***ADEPT WSM-Nets*** [12, 32, 40].

Généralement, dans les techniques se basant sur la définition de **critères de correction à partir d'une équivalence de traces d'exécution** (*Trace Equivalence*), comme les *Flow Nets*, *WIDE Graphs*, *Breeze Activity Nets*, *TRAMs Graphs* et *ADEPT WSM-Nets*, l'équivalence de traces se focalise sur l'évaluation de l'historique d'exécution de l'instance I [9, 10, 32, 11]. Si cette exécution a pu être potentiellement réalisée selon un schéma S', une migration de I vers S' est alors envisagée. De plus, les *Flow Nets* [6] offrent une **approche prédictive**, prenant en compte l'exécution future de l'instance selon le schéma transformé.

WIDE [9] offre, par exemple, **un ensemble complet et minimal d'opérations** basiques pour transformer un schéma correct S en un autre schéma correct S' (voir section 1.1.4). Pour migrer les instances vers le nouveau schéma S', une propriété de conformité restrictive (voir section 2.2) est utilisée pour vérifier si la réalisation complète de l'historique d'une instance I selon un schéma S pourrait également être produite par l'exécution de cette même instance selon le schéma modifié S'.

TRAM se fonde sur les concepts de la gestion des versions des schémas. Pour gérer efficacement la migration d'une instance, les auteurs proposent une définition d'une **condition de migration** (*Migration Condition*) pour chaque opération de changement qui est, à quelques différences près, semblable à la **propriété de conformité globale** de [32]. Avec ces conditions, il possible de trancher si une instance peut progressivement migrer vers la nouvelle version du schéma.

Pour mieux comprendre les différences entre les approches pour résoudre ces « problèmes causés par le changement dynamique», il est nécessaire de s'intéresser à la manière dont celles-ci définissent leur critère de correction. Si on considère qu'un changement est représenté par une déviation Δ, admettant

que S soit le schéma du Workflow, et I soit une instance de S. Supposons que S soit transformé en un autre schéma correct S' en appliquant le changement Δ. La signification de la correction du schéma dépend des propriétés structurelles et du critère de correction du métamodèle de Workflow utilisé. Nous employons le terme « **migration** », citer dans [3], pour designer le passage du modèle de l'instance I d'un schéma S à un schéma S'. Veuillez vous reporter à l'annexe E pour une formalisation détaillée.

Certaines approches, comme les *WF Nets*, *MILANO Nets* et $WASA_2$ *Activity Nets*, fondent leurs critères de correction sur une **équivalence de graphes du Workflow** (le mot « graphes» est employé pour généraliser les typologies des réseaux de représentation graphiques utilisées par les différentes approches). Le principe fondamentale de cette équivalence de graphes est, soit de comparer les schémas du Workflow avant et après l'occurrence du changement [27], ou soit d'essayer de lier le graphe d'une instance I au schéma transformé S' [8, 4]. Un **degré de couverture** permet de décider si le changement Δ est applicable ou non à l'instance I.

Au détriment d'une certaine facilité de modélisation, les approches fondées sur les réseaux de Petri (c'est-à-dire, fondées sur une **sémantique de *jetons-vrai/faux***) souffrent de plusieurs problèmes inhérents à cette modélisation. Souvent, il leur manque une séparation claire entre les jetons représentant le flot de contrôle, de ceux du flot de données, ce qui, inévitablement complique le changement (dynamique) dans ce genre de réseau. Dans *Flow*, le schéma du Workflow et ses instances sont représentés dans un même réseau de Petri avec un marquage coloré. Toutefois, chaque instance possède son propre marquage défini sur ce schéma. Une modification du schéma consiste alors en une substitution des sous-réseaux marqués. Le maintien de la cohérence du nouveau schéma passe par une vérification de la topologie du réseau occultant, cependant, un contrôle de la conformité des instances avec le nouveau réseau sous des conditions formelles [41] reste nécessaire.

En plus, un autre problème apparaît du fait que la distinction n'est pas explicite entre le marquage des régions antérieurement exécutées (car, il n'est pas conservé) et celui des régions « sautées» (du fait, qu'elles ne sont pas du tout marquées). Par conséquent, la question est, alors, de savoir comment adapter le marquage des instances après la propagation d'un changement du schéma sans connaissance préalable de leurs exécutions antérieures. Dans [39], l'administrateur du Workflow se doit d'adapter manuellement le marquage pour chaque instance. Ainsi, une analyse complexe est nécessaire pour vérifier si toutes les activités résultantes (devant êtres exécutées après l'occurrence du changement) sont atteignables, garantissant en conséquence la cohérence des instances après une modification du marquage. De plus, les réseaux de Petri ont un sérieux handicape (implicite) à modéliser les cycles (voir section 1.1.5). Ainsi, la distinction entre les cycles souhaités dans le procédé et les cycles provoquant des interblocages reviendrait à résoudre un problème *NP-complet*. En ce sens, les travaux [42, 37, 32] proposent un critère de conformité [32] et des algorithmes d'adaptation [42, 37] d'une complexité linéaire.

L'adaptation des marquages est perçue comme un problème très complexe : c'est « le bogue du changement dynamique». Pour corriger ce bogue, les auteurs suggèrent que l'administrateur spécifie le marquage du nouveau réseau (obtenu après modification) qui doit être applicable pour chaque instance, et ce, en relation avec le marquage de l'ancien réseau [27].

D'autres approches plus récentes se sont confrontées à cette problématique. Dans [7], les auteurs proposent de ne pas propager les modifications du schéma du Workflow aux instances qui exécutent des activités se trouvant dans la région modifiée. Une **analyse du delta du changement** [40] (*Delta Analysis*), matérialisée par une comparaison structurelle basée sur les **relations d'héritage** [43] qui existent entre le schéma et les instances (la relation d'équivalence définie un ordre partiel sur le cycle de vie d'objets), permet de déterminer le **périmètre de la modification**. Cette relation d'héritage, ainsi que les définitions de la **dissemblance** (*Disjointness*) et du **chevauchement** (*Overlapping*) introduites dans [40] se basent sur les notions d'équivalence entre les schémas de Workflow. Tandis que *V.d. Aalst* et *Basten* utilisent la **bissimilarité des ramifications** (*Branching Bisimilarity*) comme relation d'équivalence préservant la structure en réseaux des modèles de Workflow [43, 27, 44], il existe bien d'autres notions d'équivalence entre les modèles d'un Workflow, par exemple, celle de l'**isomorphisme des graphes** de schémas [24]. Outre l'analyse du delta du changement qui propose des méthodes intéressantes pour maintenir le sens sémantique du schéma du procédé avant et après le changement en appliquant des transformations préservant la sémantique. Dans [40], les auteurs proposent un *framework* formel pour

faire face à des changements concurrents du procédé (c'est-à-dire, aussi bien au niveau du schéma, qu'au niveau des instances du procédé). Une application importante de ce résultat est la possibilité de propager des changements du schéma à des **instances dites « biaisées »** (ayant dérivée du schéma initial par des changements individuels d'une manière *ad-hoc*). Ainsi, en utilisant un degré particulier de correspondance entre les changements du schéma et des instances, il est possible de choisir différentes stratégies de migration (il s'agit de la migration unilatérale des modification du schéma vers les instances). Pour déterminer ce degré, ils proposent pour cela 3 méthodes : structurelle, opérationnel, hybride [40, 23].

En résumé, toutes ces approches sont trop restrictives vis-à-vis de la tolérance aux boucles car elles sont fondées sur le respect de propriétés structurelles restrictives. Dans ce sens, [32] tente d'y remédier en offrant un modèle et des opérations de changements utilisant un **critère de conformité plus compréhensif**. Par ailleurs, elles n'examinent pas l'impact de l'évolution par rapport à l'aspect du flot de données dans les schémas de Workflow. Enfin, les auteurs ne montrent pas comment les critères de correction suggérés peuvent être formellement vérifiés, ce qui est souvent important quand on intègre une vérification de conformité dans une implémentation d'un moteur d'exécution de procédés. Des approches orientées objets sont présentées par *Joeris* et *Herzog* [45] et *Weske* [33]. Dans *MOKASSIN* [45], on retrouve une **notion de correction définie par les concepteurs**, les changements sont effectués par encapsulation des primitives de changement au sein des instances. Par conséquent, les instances du procédé ou les utilisateurs sont eux-mêmes responsables de la préservation de la cohérence. Le caractère de la propriété de conformité est considéré comme étant trop restrictif, un concept de **gestion des versions plus granulaire** est proposé, sans pour autant soulever les questions liées à la vérification (efficace) de cette conformité. Une autre approche de gestion de versions a été présentée par $WASA_2$ [33], proposant une vérification efficace de conformité en établissant un lien entre le schéma modifié et les sous-Workflows résultants des instances correspondantes.

Le prototype $ADEPT_2$ [46] propose un **critère indépendant du métamodèle du Workflow utilisé**. Ce critère se fonde sur une notion moins restrictive de l'équivalence des traces d'exécution [32]. L'attention est portée aussi bien sur les changements du flot de contrôle que ceux du flot de données. Le critère permet une tolérance au problème de boucles excluant les conflits structurels et sémantiques [47] (voir section 1.1.4).

2.4 Conclusion

Plusieurs métamodèles ont étés proposés dans la littérature comme prémisses au changement dans les Workflows, en comparant ces formalismes il est évident de trouver de nombreuses différences. Par exemple, seules les *WF Nets*, *Flow Nets*, *WIDE Graphs*, et *ADEPT WSM-Nets* permettent la modélisation des boucles (voir section 1.1.5). La modélisation du flot de données n'est prise en compte que dans les *WF Nets* et les *MILANO Nets*. *WIDE* ne journalise que des entrées de l'achèvement des activités, alors que l'historique d'exécution dans *TRAMs* et *ADEPT* stocke, en plus, les entrées de démarrage des activités.

La pertinence d'un modèle se juge essentiellement par l'application à laquelle il est destiné. Ainsi, toutes ces approches abordent les problèmes liés aux **changements dynamiques** dans les Workflows. Cependant, à un niveau plus abstrait, une vue d'ensemble des propriétés d'une approche est essentielle pour pouvoir répondre à l'interrogation : qu'est-ce qu'est censé apporter un modèle de Workflow ? Le lecteur intéressé pourra se reportait à [12] pour une comparaison détaillée selon l'expressivité du critère de correction qu'elles définissent, et une confrontation des techniques utilisées pour résoudre les problèmes inhérents à la flexibilité des systèmes de gestion de Workflow.

Dans cet état de l'art, nous avons pu voir que la littérature propose une multitude de solutions basées sur les états d'avant et d'après le changement pour résoudre la problématique du changement dynamique dans les Workflows. Sauf que la complexité accrue de ces solutions pose de nombreux soucis d'ingénierie lorsqu'il s'agit de les mettre en œuvre. Outre la difficulté de leurs exploitations, la traçabilité, le suivi, l'analyse et la réutilisation des changements sont considérablement limitées. Face à ce constat, nous présentons dans le chapitre suivant une approche innovante basée sur les opérations pour la gestion de l'évolution du schéma et du changement dynamique des instances en cours d'exécution.

Chapitre 3

Propositions et réalisations

Sommaire

Dans ce chapitre nous allons énoncer les principales composantes du *framework* que nous proposons. L'objectif de cet énoncé est de présenter la formalisation de notre approche pour la gestion de l'évolution des spécifications d'un modèle de Workflow. L'enjeu, ici, est ni de présenter un ensemble complet et minimal d'opérations de changement qui se distingue des approches précédentes, ni une technique pour gérer les liens entre les versions des schémas et des instances [48, 49], mais de concevoir un nouveau paradigme du Workflow afin d'améliorer la flexibilité et l'adaptabilité des systèmes de gestion des procédés métiers. D'abord, nous présentons notre *framework* formel basé sur un paradigme de systèmes d'opérations- *Stratégie/Incidence(s)*- offrant un cadre flexible pour effectuer des changements, afin de faire évoluer les modèles de Workflow tout en préservant leur cohérence d'exécution. Cette proposition représente le cœur de notre *framework*. De part sa généricité, elle permet de répondre au besoin de gérer aussi bien l'évolution du modèle de workflow, que d'offrir les mécanismes nécessaires pour effectuer des changements dynamiques de l'exécution, et de pouvoir les réutiliser. Ensuite, nous définissons un critère de correction permettant de contrôler l'impact des changements sur la structure d'un Workflow afin de préserver la cohérence des exécutions. Comme l'évolution des modèles ou les changements in situ des exécutions reflètent un savoir-faire et nécessitent une expertise, il nécessaire de pouvoir capitaliser le changement en proposant des techniques pour leur réutilisation. Nous définirons alors un graphe des arrangements lexicographique d'opérations centré sur le noyau dont la topologie permet le suivi et la traçabilité des opérations de changement. Finalement, nous étudierons la complexité de notre approche vis-à-vis des solutions existantes, qui se basent sur les états d'avant et d'après le changement pour résoudre la problématique des modifications dans les Workflows.

3.1 Objectifs

Grâce à l'étude des différentes solutions pour l'évolution du Workflow et la gestion du changement dynamique, nous nous sommes rendu à l'évidence que toutes ces solutions étaient basées sur l'évaluation des états- avant est après l'occurrence du changement -du modèle de Workflow pour définir leur critère de

conformité ou de correction. Ainsi, pour que cette évaluation des états puisse être possible, et par exemple les approches [45, 10, 46], se retrouvent à gérer les versions du schéma matérialisant son évolution, d'une part. Et d'autre part, à maintenir les liens de dépendances entre les instances et les différentes versions du schéma. Par la conservation des versions, synonymes de l'historique des états du schéma, cette technique devient onéreuse en terme d'utilisation de l'espace mémoire et de complexité des traitements, lors de la présence d'un grand nombre de versions du schéma. De ce fait, une modification mineure conduit à la création d'une nouvelle version du schéma [10]. D'un autre côté, le changement *ad-hoc* des instances est souvent relégué au second plan, car il est difficile, dans bien de cas (difficulté à définir un métamodèle), de maintenir aussi l'historique de leurs modifications. Le problème est alors d'identifier selon quel schéma s'exécute une instance biaisée. Est-ce une évolution de son schéma initial, et donc il s'agirait de gérer des versions du schéma ? Ou est-ce un schéma totalement indépendant du schéma initial, et dans ce cas il s'agit d'évaluer l'équivalence entre schémas ? Le problème devient plus critique encore, lorsqu'il s'agit d'évaluer des classes d'instances.

Devant ce constat, nous nous sommes adonné aux réflexions suivantes : *(i)* Étant donné que le changement représente en lui-même une opération (car on parle d'opérations ou de primitives de changement, voir la section 1.1.4), et plus encore, la construction d'un modèle de Workflow n'est en réalité qu'une suite d'opérations de changement, initiées sur un schéma vide, et ensuite constituées d'insertions et de suppressions de composants du Workflows (des activités, des données, *etc.*). Ce qui, évidemment, n'est pas considéré par les approches existantes, qui ne s'intéressent au concept de changement que dès qu'il s'agit de remettre en cause un modèle existant, nécessairement déployé et instantié par différentes exécutions. *(ii)* Et si c'est le paradigme même des workflows, faisant qu'une instance doit toujours être liée (et plus encore, conforme) à un schéma, qui est à l'origine de la complexité à gérer les liaisons entre les versions du modèle du schéma et celle des instances en exécution. Subséquemment, considérer l'évolution d'un modèle de Workflow, dès sa création, comme un enchaînement d'opérations, permet-il d'utiliser les techniques de la réplication optimiste (en particulier les transformées opérationnelles) et celle de la réconciliation basée sur les opérations. Cependant, il faut dire que l'objectif ici n'est pas le même, car il ne s'agit pas de converger vers un état global commun entre le schéma et les instances. Mais néanmoins, il nous est possible de s'inspirer de ces techniques, puisque leur logique permet de reproduire des opérations se produisant sur une entité différente de celle où il est nécessaire de l'intégrer.

3.2 Gestion du changement

Un des concepts-clés de notre approche est le système d'opération. Comme cité précédemment, celui-ci permet d'élever le paradigme « ***Schéma/Instance(s)*** » d'un WfMS à un ensemble d'opérations. Ce concept est défini de la manière suivante :

Définition ***(Système d'opérations)***. Un système d'opérations est un triplet $SO = (OP^{SO}, HL^{SO}, R^{SO})$ où,

- $OP^{SO} = OP^{SO}_{Cycledevie} \cup OP^{SO}_{Changement}$ est un ensemble d'opérations, où
  - $P^{SO}_{Cycledevie}$ est un ensemble d'opérations de cycle de vie ;
  - $OP^{SO}_{Changement}$ est un ensemble d'opérations de changement.
- HL^{SO} est l'horloge logique de SO ;
- R^{SO} représente la relation d'ordre (*précédence causale*) définie par les estampilles des opérations OP.

Dans la suite, nous utiliserons la notation $[op_i]$ comme un tuple pour désigner un ensemble ordonnées d'opérations $\{op_1, ..., op_n\}$.

La relation d'ordre sur les opérations requiert de doter tout système d'opérations d'une horloge logique qui permet d'estampiller chacune des opérations se produisant au sein même du système. Le rôle de cet estampillage est double. D'une part, il est possible d'ordonner deux opérations se produisant sur deux systèmes d'opérations distincts. D'autre part, l'estampille représente un identifiant unique et absolu permettant de déterminer l'appartenance d'une opération à son système. L'estampille d'une opération

(se reporter à la définition dans C) représente ainsi son identifiant dans un système d'opération, ce qui permet de définir une relation d'ordre entre toutes les opérations d'un système d'opération. Pour alléger notre présentation, on utilisera la notation op_i pour désigner la $i$ème opération d'un système d'opération. Une opération est définie comme suit :

Définition ***(Opération)***. Une opération est un tuple $op = (f^{op}, D^{op}, T^{op}, SO^{op}, HL(op), P^{op}, R^{op}, C*^{op})$ où,

- f^{op} est une fonction n-aire sur son domaine de définition D^{op} ;
- $D^{op} : f^{op} \times PCG \mapsto \{Activités, Transitions, (Données, ...)^{5}\} \cup \{\text{Système d'opérations}\}$ est son domaine de définition qui dépend de la politique de changement globales (voir plus loin) ;
- $T^{op} : f^{op} \mapsto \{Changement, Cycledevie, Projection, Intégration\}$ est son type ;
- SO^{op} désigne le système d'opération auquel elle appartient ;
- $HL(op)$ représente son estampille dans le SO^{op} ;
- $P^{op} : f^{op} \mapsto \{Globale, Privée\}$ désigne sa portée ;
- $R^{op} : f^{op} \mapsto \{Vrai, Faux\}$ désigne le résultat qu'elle retourne ;
- $C*^{op} : f^{op} \mapsto \{Instantanée, Diff érée\}$ est une condition (optionnelle) d'intégration de l'opération, qui peut être structurelle (exemple, l'insertion d'une activité ne peut avoir lieu qu'après la suppression d'une autre activité) ou événementielle (exemple, l'insertion d'une activité ne peut avoir lieu qu'après l'occurrence d'un certain événement).

Selon nos hypothèses, on ne s'intéresse ici qu'à des opérations de flot de contrôle et de cycle de vie, et dans un soucis de brièveté, l'évaluation de la condition de l'application de l'opération est hors de nos propos. Notre volonté de spécifier un *framework* générique nous amène à définir la notion d'objets d'une opération, et en conséquence, la définition d'une activité dans un flot de contrôle que voici :

Définition ***(Objet)***. Un objet est un concept manipulé par la fonction d'une opération. Ce concept est, dans notre cas précis, un objet de flot de contrôle, dans le sens où les opérations manipulent des activités.

Définition ***(Activité)***. Une activité est un objet appartenant au domaine de définition des opérations du changement et du cycle de vie. Elle est définie par le tuple $A = (APred^{A}, ASucc^{A}, TPred^{A}, TSucc^{A}, E^{A}, T^{A})$ où,

- $APred^{A}$ est l'ensemble des activités précédents A ;
- $ASucc^{A}$ est l'ensemble des activités successeurs de A ;
- $TPred^{A}$ est l'ensemble des transitions en amont de A ;
- $TSucc^{A}$ est l'ensemble des transitions en aval de A ;
- $E^{A} : A \mapsto \{NonActivée(), Activée(), Abandonnée(), Exécutée(), Terminée()\}$ décrit les primitives d'état de A ;
- $T^{A} : A \mapsto \{Initiale, Finale, Séquentielle, Concurrence, Imbriquée, Itérative, Sélection, Synchronisation, Réunion\}$ décrit le type de A.
- *Cas particuliers* :
  - $T^{A} : \{Initiale\} \mapsto \{NonActivée(), Abandonnée(), Activée()\}$;
  - $T^{A} : \{Finale\} \mapsto \{NonActivée(), Abandonnée(), Terminée()\}$.

Pour le cas des activités initiales et finales, on a : $(T^{A} = \{Initiale\} \Leftrightarrow TPred^{A} = \emptyset)$ et $(T^{A} = \{Finale\} \Leftrightarrow TSucc^{A} = \emptyset)$.

Puisqu'on ne s'intéresse qu'à des changements du flot de contrôle dans un Workflow, nous définissons les opérations qui suivent. Pour mieux comprendre la hiérarchie des opérations, les figures A.13 et A.14 de l'annexe A expose un diagramme de classes UML de la structure de notre *framework*.

[5] *Il est possible d'étendre les opérations à tout type d'objets contenus dans un modèle de procédé.*

Définition ***(Opération de changement du flot de contrôle d'un système d'opérations)***. Une opération de changement du flot de contrôle $op_{Changement}$ est une fonction qui manipulent des objets de type activité et un système d'opération. Elle peut avoir une portée globale ou privée.

- $InsérerSérie(so : SO; a_{ins}, a_{pré}, a_{post} : A)$ insère l'activité a_{ins} entre les activités $a_{pré}$ et a_{post} au sein du système d'opérations so ;
- $InsérerParallèle(so : SO; a_{ins}, a_{pré}, a_{post} : A)$ insère l'activité a_{ins} entre les activités $a_{pré}$ et a_{post}, parallèlement aux activités $\{\forall a, b/a \in ASucc^{a_{pré}} \wedge b \in APred^{a_{post}}\}$, au sein du système d'opérations so ;
- $Supprimer(so : SO; a_{sup} : A)$ supprime l'activité a_{sup} du système d'opérations so.

Définition ***(Opération de cycle de vie d'un système d'opérations)***. Une opération de cycle de vie $op_{Cycledevie}$ d'un système d'opérations est une fonction qui manipule d'autres opérations ou des activités. Elle ne peut avoir qu'une portée privée (voir l'algorithme B.1 de l'annexe B).

- $Créer(so : SO; a : A)$ crée l'activité a au sein du système d'opérations auquel appartient l'opération et met l'activité dans l'état $NonActivée()$;
- $Créer(so : SO; t : T; a_{pré}, a_{post} : A)$ crée une transition t entre les activités $a_{pré}$ et a_{post}, et positionne la transition à $NonActivée()$;
- $Créer(so : SO; [op_i] : ops)$ crée un système d'opérations avec les opérations de changement $[op_i]$ lors d'une exportation ;
- $Activer(so : SO)$ active le système d'opérations auquel elle appartient avec la visibilité adéquate, définie selon la politique d'exportation ;
- $Désactiver(so : SO)$ désactive le système d'opérations auquel elle appartient pour qu'il ne puisse pas être exporté.

Ces opérations sont communes aux stratégies ainsi qu'aux incidences du flot de contrôle d'un procédé, que nous présentons plus loin. Pour alléger nos propos, nous ignorerons volontairement de citer les opérations manipulant les transitions entre les activités.

Définition ***(Opération de projection d'un système d'opérations)***. Chaque opération de projection s'effectue en quatre étapes : *(i)* elle est intégrée localement au système d'opération, dans le cas où l'intégration réussie, *(ii)* elle est notifiée aux autres systèmes d'opérations, *(iii)* elle est reçue par les autres systèmes d'opérations, *(iv)* et y est (ou non) intégrée. L'opération de projection à une portée privée au système d'opération.

- $Exporter(so, so_{nouveau} : SO)$ permet de créer un système d'opérations $so_{nouveau}$, et d'exporter les opérations de changement globales du système d'opération so auquel elle appartient vers $so_{nouveau}$ (la figure A.4 de l'annexe A en donne un exemple) ;
- $Notifier(so, so_{expéditeur} : SO; [op_i] : ops)$ (voir l'algorithme B.5 dans l'annexe B) notifie les systèmes d'opérations SO^{so} de so de l'occurrence des opérations globales $[op_i]$ dans le système d'opérations $so_{expéditeur}$ (Pour une illustration, voir figure A.5 de l'annexe A).

L'opération d'exportation permet ainsi une plus grande flexibilité pour la réutilisation des opérations de changement. Elle offre la possibilité de construire un nouveau système d'opérations SO^1 (une stratégie ou une incidence) avec un ensemble d'opérations contenu dans un autre système SO^2. Le système d'opérations SO^1, ainsi créé, est une dérive de SO^2. Réciproquement, SO^2 est le noyau de SO^1. Par conséquent tout changement global se produisant dans SO^1 est notifié à SO^2, et inversement. Par contre, les opérations de changement locales de SO^2 ne sont pas notifiées à SO^1.

Définition ***(Opération d'intégration d'un système d'opérations)***. La notification utilise cette opération d'intégration pour intégrer les opérations aux systèmes d'opérations (voir les algorithmes B.8 et B.9 de l'annexe B) :

- $Intégrer(so : SO, op)$ intègre op à SO.

Les algorithmes de toutes ces opérations se trouvent en annexe B, et les figures A.4 et A.5 de l'annexe A montre des exemples d'évolution de systèmes d'opérations.

3.3 Traçabilité du changement

Avant d'introduire le paradigme de « ***Stratégie/Incidence(s)*** », il est intéressant de définir les concepts de politique et de directives de changement qui permettront de saisir la généricité de notre modèle par rapport à toute politique de changement décrite dans la section 1.1. Explicitement, cela permettra au WfA, à travers une formulation des directives, de mettre en œuvre une politique de gestion du changement au sein du WfMS.

Définition ***(Politique et directives du changement)***. Une politique du changement est un tuple $P = (D^P, P_{PCG}, P_{PCT}, P_{PCI}, P_{PCP}, P_{PCE}, P_{PCEGP}, P_{PCR})$ où,
- D^P est un ensemble de directives ;
- $P_{PCG} : D^P \mapsto \{Flotdecontrôle, (Flotdedonnées, ...)^6\}$ est la politique de changement globales ;
- $P_{PCT} : D^P \mapsto \{Instantané, Diffé ré\}$ est la politique des types de changements des opérations ;
- $P_{PCI} : D^P \mapsto \{AvecAdhoc, SansAdhoc\}$ est la politique du changement pour la dérive des incidences ;
- $P_{PCP} : D^P \mapsto \{AvecPublication, SansPublication\}$ est la politique de publication des opérations par les incidences ;
- $P_{PCE} : D^P \mapsto \{Avec/SansExportation, AvecExportation\}$ est la politique d'exportation des changements dans les systèmes d'opérations ;
- $P_{PCEGP} : D^P \mapsto \{GExportation, PExportation, Null\}$ est la politique d'exportation des changements dans les systèmes d'opérations ;
- $P_{PCR} : D^P \mapsto \{AvecRejet, SansRejet\}$ est la politique d'exportation des changements rejetés dans les systèmes d'opérations.

Comme notre proposition se veut ne pas être tributaire d'un WfMS particulier, et pour les besoins de son implémentation, deux choix se présentent. Le premier est de développer un nouveau WfMS centré sur notre *framework*. Le second est d'intégrer ce *framework* dans un WfMS existant, et dans ce cas, il est nécessaire de prendre en considération le paradigme *Schéma/Instance(s)* pour éviter la réingénierie de toute l'architecture du WfMS. Voici donc une définition d'un schéma et d'une instance (à un haut niveau d'abstraction) au sens classique d'un moteur d'exécution.

Définition ***(Schéma du flot de contrôle)***. Un schéma du flot de contrôle est un couple $SFC = (A^{SFC}, TA^{SFC})$ où,
- A^{SFC} est un ensemble d'activités ;
- TA^{SFC} est une relation représentant les transitions de dépendance de contrôle entre les activités A^{SFC}.

Définition ***(Instance du flot de contrôle)***. Une instance du flot de contrôle est un tuple $IFC = (SFC^{IFC}, VT^{SFC})$ où,
- SFC^{IFC} est le schéma du flot de contrôle selon lequel s'exécute IFC ;
- $VT^{SFC} = (EA^{IFC}, ET^{IFC})$ est une relation décrivant les opérations sur les activités et les transitions de IFC :
  - $EA^{IFC} : A^{SFC^{IFC}} \mapsto \{NonActivée(), Activée(), Abandonnée(), Excutée(), Terminée()\}$;
  - $ET^{IFC} : TA^{SFC^{IFC}} \mapsto \{NonActivée(), Activéeà Vrai(), Activéeà Faux()\}$.

Par rapport à un schéma du flot de contrôle, une instance peut être vue comme un schéma avec des propriétés d'exécution sur les activités. Dans le but d'alléger la présentation, on ne considérera que les opérations manipulant les activités. Ainsi l'état d'une activité détermine le type des transitions directes qui se trouvent en aval de cette activité. Et par conséquent, le type des transitions en amont d'une activité

[6] *Il est possible d'étendre les changements à tout type de flot.*

détermine les primitives de son état. Considérons une Activité $a \in A^{SFC^{IFC}}$ d'une instance du flot de contrôle IFC :

- si $\forall t \in TPred^a \wedge t = \{NonActivée()\} \Rightarrow E^a = \{NonActivée()\}$;
- si $\forall t \in TPred^a \wedge t = \{ActivéeàFaux()\} \Rightarrow E^a = \{Abandonnée()\}$;
- si $\forall \in TPred^a \wedge t = \{ActivéeàVrai()\} \Rightarrow E^a = \{Activée()\}$;
- si $E^a = \{NonActivée(), Activée(), Excutée()\} \Rightarrow \forall t \in TSucc^a \wedge t = \{NonActivée()\}$;
- si $E^a = \{Abandonnée()\} \Rightarrow \forall t \in TSucc^a \wedge t = \{ActivéeàFaux()\}$;
- si $E^a = \{Terminée()\} \Rightarrow \forall t \in TSucc^a \wedge t = \{ActivéeàVrai(), ActivéeàFaux()\}$. Cela dépendra de la condition d'activation de la transition contenue dans le code de l'activité.

L'objectif principal de notre approche étant de proposer un nouveau paradigme pour les WfMS qui permet une meilleure flexibilité de la gestion du changement, c'est ainsi que nous définissons les deux concepts de stratégie et d'incidence(s).

Définition ***(Stratégie du flot de contrôle d'un procédé)***. Une stratégie du flot de contrôle d'un procédé est un système d'opérations définit par un tuple $S = (OP^S, SO^S, SFC^S, V^S)$ où,

- $OP^S = (OP^S_{Privées} \cup OP^S_{Globales} \cup OP^S_{Rejetées} \cup OP^S_{Différées})$ décrit l'ensemble des opérations de la stratégie S tels que :
  - $OP^S_{Privées} = OP^S_{Cycledevie}$ désigne les opérations privées définies sur S, et qui représentent des opération de cycle de vie locales :
    - $OP^S_{Cycledevie} : op^S \mapsto \{Activer(S)$, $Désactiver(S)$, $Créer(S, [op_i])$, $Créer(S, a^{SFC^S})$, $Créer(S, t^{SFC^S})$, $Exporter(S, so_{nouveau})$, $Notifier(S, SO_{expéditeur}, [op_i])$, $Intégrer(S, op)\}$.
  - $OP^S_{Globales} = OP^S_{Changement}$ désigne les opérations globales définies sur S, et qui représentent des opération de changement devant être projetés sur les SO^S :
    - $OP^S_{Changement} : op^S \mapsto \{InsérerSérie(S, a_{ins}, a_{pré}, a_{post})$, $InsérerParallèle(S, a_{ins}, a_{pré}, a_{post})$, $Supprimer(S, a_{sup})\}$.
  - $OP^S_{Rejetées}$ désigne les opérations qui n'ont pas pu être intégrées à S ;
  - $OP^S_{Différées}$ désigne les opérations qui n'ont pas étés intégrées, mais qui sont différées, et qui doivent êtres intégrées dès que leurs conditions d'intégration sont satisfaites ;
- $SO^S = (SO^S_{Noyau}, SO^S_{Dérives})$ est l'ensemble des systèmes d'opérations liés à S :
  - $SO^S_{Dérives}$ est l'ensemble des systèmes d'opérations dérivant de S (c'est-à-dire, qui son proportionnelles à S) ;
  - SO^S_{Noyau} est le système d'opérations dont dérive S. Il peut être vide, et dans ce cas la stratégie est dite originale ;
- SFC^S est le schéma du flot de contrôle sur lequel S est appliquée ;
- $V^S : OP^S \times P^S_{PCE} \mapsto \{Exportable, NonExportable\}$ décrit la validité de S, à savoir si elle est exportable (c'est-à-dire, d'autres systèmes d'opérations peuvent en dérivés ou non) :
  - $V^S : OP^S \times \{Avec/SansExportation\} \mapsto \{Exportable, NonExportable\}$;
  - $V^S : OP^S \times \{AvecExportation\} \mapsto \{Exportable\}$.

La stratégie, en particulier le cas d'une stratégie du flot de contrôle, représente le plan de conduite de la construction d'un procédé, et par conséquent le plan de conduite du changement. Ce plan de conduite du changement est décrit dans un schéma de flot de contrôle SFC^S. Se reporter à la figure A.3 de l'annexe A pour une illustration de l'analogie entre l'évolution d'une stratégie et de son schéma.

Définition ***(Incidence du flot de contrôle d'un procédé)***. Une Incidence du flot de contrôle d'un procédé est un système d'opérations défini par un tuple $I = (OP^I, SO^I, IFC^I, V^I)$ où,

- $OP^I = (OP^I_{Privées} \cup OP^I_{Globales} \cup OP^I_{Rejetées} \cup OP^I_{Différées})$ décrit l'ensemble des opérations de l'incidence I tels que :

- $OPI^I_{Privées} = (OP^{(I)Privées}_{Cycledevie} \cup OP^{(I)Privées}_{Changement})$ désigne les opérations privées définies sur I. Les $OP^{(I)Privées}_{Changement}$ peuvent être publiées, et dans ce cas elles doivent être projetées sur SO^I ;
- $OPI^I_{Globales} = (OP^{(I)Globales}_{Cycledevie} \cup OP^{(I)Globales}_{Changement})$ désigne les opérations globales définies sur I, et qui doivent être projetés sur SO^I :
  - $OPI^I_{Changement} = (OP^{(I)Globales}_{Changement} \cup OP^{(I)Privées}_{Changement})$:
    - $OPI^I_{Changement} : op^I \mapsto \{InsérerSérie(I, a_{ins}, a_{pré}, a_{post}), InsérerParallèle(I, a_{ins}, a_{pré}, a_{post}), Supprimer(I, a_{sup})\}$;
  - $OPI^I_{Cycledevie} = OP^{(I)Privées}_{Cycledevie}$:
    - $OPI^I_{Cycledevie} : op^I \mapsto \{Activer(I)$, $Désactiver(I)$, $Créer(I, [op_i])$, $Créer(I, a^{IFC^I})$, $Activer(I, a^{IFC^I})$, $Abandonner(I, a^{IFC^I})$, $Exécuter(I, a^{IFC^I})$, $Terminer(I, a^{IFC^I})$, $Exporter(I, so_{nouveau})$, $Notifier(I, [op_i])$, $PublierLocalement(I, [op_i])$, $Notifier(I, SO_{expéditeur}, [op_i])$, $PublierGlobalement(I, [op_i])$, $Intégrer(I, op)$, $ActiverVrai(I, t^{IFC^I})$, $ActiverFaux(I, t^{IFC^I})$, $Déactiver(I, t^{IFC^I})$, $Créer(I, t^{IFC^I})\}$;
  - $OPI^I_{Rejetées}$ désigne les opérations qui ne peuvent être définitivement intégrées à I ;
  - $OPI^I_{Différées}$ désigne les opérations qui n'ont pas été intégrées, et qui sont différées, mais devant êtres intégrées dès que leurs conditions d'intégration sont satisfaites ;
- $SO^I = (SO^I_{Noyau}, SO^I_{Dérives})$ est l'ensemble des systèmes d'opérations liés à I :
  - $SO^I_{Dérives}$ est l'ensemble des systèmes d'opérations dérivant de I ;
  - SO^I_{Noyau} est le système d'opérations dont dérive I. Il peut être vide, et dans ce cas l'incidence est dite originale ;
- IFC^I est l'instance du flot de contrôle sur laquelle est appliquée I ;
- $V^I : OP^I \times P^I_{PCE} \times P^I_{PCEGP} \mapsto \{GlobalesExportable, PrivéesExportable, NonExportable\}$ décrit la validité de I, à savoir si elle est exportable (c'est-à-dire, d'autres systèmes d'opérations peuvent en dérivés ou non) :
  - $V^I : OP^I \times \{Avec/SansExportation\} \times \{GExportation\} \mapsto \{GlobalesExportable, NonExportable\}$;
  - $V^I : OP^I \times \{Avec/SansExportation\} \times \{PExportation\} \mapsto \{PrivéesExportable, NonExportable\}$;
  - $V^I : OP^I \times \{AvecExportation\} \times \{Null\} \mapsto \{GlobalesExportable, PrivéesExportable\}$.

Cette énonciation a pour objectif de mettre en évidence le contenu d'un système d'opérations. Nous détaillons ici les notions de noyau et de dérives composant l'ensemble SO^{so} d'un système d'opérations *so*.

Définition ***(Noyau et dérives d'un système d'opérations)***. Un système d'opération SO^1 est dit noyau de SO^2, si SO^2 a été créé par exportation des opérations de changement de SO^1. SO^2 est alors une dérive (ou dérivé) de SO^1. Un système d'opération SO peut avoir plusieurs dérives, notées $SO^{SO}_{Dérives}$, mais ne peut posséder qu'un seul noyau, noté SO^{SO}_{Noyau}.

Il est évident que certaines opérations définies précédemment doivent êtres redéfinies pour les propriétés particulières d'une incidence. Les tableaux B.3 et B.4 de l'annexe B énoncent les algorithmes de l'opération d'exportation pour une stratégie et une incidence. Se reporter à la figure A.14 de l'annexe A pour un diagramme de classes UML spécifiant la hiérarchie des opérations du *framework*.

Définition ***(Opération de cycle de vie d'une incidence)***. Une opération de cycle de vie $op_{Cycledevie}$ d'une incidence I est une opération qui manipule, en plus, les activités de l'instance du flot de contrôle IFC^I. Elle ne peut avoir qu'une portée privée (voir l'algorithme B.2 de l'annexe B).

- $Activer(in : I; a : A)$ active l'activité a ;
- $Abandonner(in : I; a : A)$ abandonne l'activité a ;
- $Exécuter(in : I; a : A)$ exécute l'activité a ;
- $Terminer(in : I; a : A)$ termine l'activité a.

Les opérations $Activer(a : A)$ et $Abandonner(a : A)$ sont exclusives l'une de l'autre, c'est-à-dire, $[\forall a \in EA^{I^{IFC}} \neg((\exists op_l \in OP^I, op_l = Activer(a)) \wedge \exists(op_k \in OP^I, op_k = Abandonner(a))]$.

- $ActiverVrai(in : I; t : T)$ active à vrai la transition t ;
- $ActiverFaux(in : I; t : T)$ active à faux la transition t.

La situation suivante doit être garantie : $[\forall t \in ET^{I^{IFC}} \neg((\exists op_l \in OP^I, op_l = ActiverVrai(t)) \wedge (\exists op_k \in OP^I, op_k = ActiverFaux(t))]$. Ces opérations ne peuvent avoir lieu que dans les incidences du flot de contrôle d'un procédé. Elles produisent les primitives d'état des activités et des transitions.

Pour que le changement local se produisant au niveau des incidences puisse aussi être réutilisé, nous définissons des opérations de projection spécifiques à une incidence.

Définition ***(Opération de projection d'une incidence)***. Une incidence du flot de contrôle possède des opérations spécifiques (par rapport à une stratégie) qui lui permettent de publier ses opérations de changement privées, qu'elle effectue d'une manière « *ad-hoc* ». Une fois exportées, ces opérations deviennent globales.

- $PublierLocalement(in : I; [op_i] : ops)$ (voir l'algorithme B.6) permet de restreindre la publication des opérations de changement privées $[op_i]$ de in qu'aux dérives $SO^{in}_{Dérives}$;
- $PublierGlobalement(in : I; [op_i] : ops)$ (voir l'algorithme B.7) permet de publier les opérations de changement privées $[op_i]$ de in à tous les SO^{in}, c'est-à-dire, aussi bien vers le noyau que vers les dérives.

3.4 La correction dans la gestion du changement

Lorsqu'un système d'opérations- stratégie ou incidence - reçoit une notification pour une opération de changement de la part d'un autre système d'opérations (appartenant à une même génération- son noyau ou ses dérives) il doit pouvoir l'intégrer.

Définition ***(Génération de systèmes d'opération)***. Une génération de systèmes d'opération est constituée d'un système d'opération dit noyau, et d'un ensemble de système d'opération dérivant directement du noyau (et qui lui sont proportionnels, voir définition dans l'annexe C). Une telle genération possède un ensemble d'opérations communes définies par les opérations de changement globales du noyau. L'algorithme d'intégration (se reporter aux algorithmes B.8 et B.9) ne doit effectivement intégrer l'opération de changement que si et seulement si elle garantit la cohérence structurelle définie sur le système d'opérations. Vu la diversité du type des opérations, et le fait que les opérations de projection, ne sont utilisées que pour garder trace de la cause du changement, il est indispensable de limiter la complexité des algorithmes par les définitions suivantes :

Définition ***(Système d'opérations du flot de contrôle réduit)***. Un système d'opérations du flot de contrôle réduit, noté SOR^{SO}, est un SO telque OP^{SO} est dépourvu des opérations de projection et des opérations de cycle de vie autres que sur des activités. Le tableau C.2 montre la différence entre SOR^{SO} et SOP^{SO} par un exemple d'incidence créée à partir d'une stratégie S dont le schéma ne contient intialement qu'une seule activité X.

Définition ***(Système d'opérations du flot de contrôle purgé)***. Un système d'opérations du flot de contrôle purgé, noté SOP^{SO}, est un SOR^{SO} sans répétitions des opérations, et par élimination des opérations sur des activités qui ont été définitivement supprimées (de SFC^S qu'il s'agisse d'une stratégie S, ou de IFC^I si c'est une incidence I). Le tableau C.1 montre la différence entre SOR^I et SOP^I d'une incidence I.

Théorème ***(Correction dynamique des opérations de changement du flot de contrôle)***. Une opération de changement du flot de contrôle est dite correcte, si et seulement elle préserve le critère de correction structurelle (par analogie à [3], de l'instance IFC^I ou du schéma SFC^S) du système d'opérations sur lequel elle est appliquée (qu'il soit une incidence I ou une stratégie S). Ce critère dynamique est défini au niveau du code de l'opération elle même. Il est local à chaque système d'opération dans lequel est intégrée l'opération de changement.

- **Pour une incidence I :**
  - T_1 : L'opération $[op :: InsérerSérie(I, a_{ins}, a_{pré}, a_{post})]$ est correcte $\Leftrightarrow [(\{\text{Créer}(a_{pré})\}, \{\text{Créer}(a_{post})\} \in SOP^I) \wedge (\{\text{Créer}(a_{ins})\}, \{\text{Activer}(a_{post})\}, \{\text{Abandonner}(a_{pré})\}, \{\text{Terminer}(a_{pré})\} \notin SOP^I)]$;
  - T_2 : L'opération $[op :: InsérerParallèle(I, a_{ins}, a_{pré}, a_{post})]$ est correcte $\Leftrightarrow [(\{\text{Créer}(a_{pré})\}, \{\text{Créer}(a_{post})\} \in SOP^I) \wedge (\{\text{Créer}(a_{ins})\}, \{\text{Activer}(a_{post})\} \notin SOP^I)]$;
  - T_3 : L'opération $[op :: Supprimer(I, a_{sup})]$ est correcte $\Leftrightarrow [(\{\text{Créer}(a_{sup})\} \in SOP^I) \wedge (\{\text{Activer}(a_{sup})\}, \{\text{Abandonner}(a_{sup})\} \notin SOP^I)]$.
- **Pour une stratégie S :**
  - T_4 : L'opération $[op :: InsérerSérie(S, a_{ins}, a_{pré}, a_{post})]$ est correcte $\Leftrightarrow [(\{\text{Créer}(a_{pré})\}, \{\text{Créer}(a_{post})\} \in SOP^I) \wedge (\{\text{Créer}(a_{ins})\} \notin SOP^I)]$;
  - T_5 : L'opération $[op :: InsérerParallèle(S, a_{ins}, a_{pré}, a_{post})]$ est correcte $\Leftrightarrow [(\{\text{Créer}(a_{pré})\}, \{\text{Créer}(a_{post})\} \in SOP^I) \wedge (\{\text{Créer}(a_{ins})\} \notin SOP^I)]$;
  - T_6 : L'opération $[op :: Supprimer(S, a_{sup})]$ est correcte $\Leftrightarrow [\{\text{Créer}(a_{sup})\} \in SOP^I]$.

Après les opérations de changement InsérerSérie (a_{ins}, $a_{pré}$, a_{post}) et InsérerParallèle(a_{ins},$a_{pré}$, a_{post}) se produisant dans un système d'opérations OS, $Créer(a_{ins})$ doit être ajoutée à $OP^{OS}_{Cycledevie}$ pour rendre compte de l'insertion effective de l'activité a_{ins}. Ainsi défini, ce critère respecte bien les invariants de la structure du modèle (c'est-à-dire, des conditions sur le schéma qui doivent être satisfaites avant et après la modification [10]). La preuve de ce théorème est donnée en annexe C. Il faut dire que ce critère local est peu restrictif, car lorsqu'il faudra gérer les évolutions d'autres aspects d'un Workflow (flot de données, rôles, *etc.*), nous pourrons l'étendre à des opérations supplémentaires. Nous pouvons remarquer qu'il est défini sur un modèle d'activités sans compensation. Cependant, par une extension de la définition du modèle d'activités, par exemple, selon les modèles de procédé transactionnels [50], il est possible de définir des critères supplémentaires.

3.5 Suivi du changement

Pour avoir un contrôle suffisant sur l'évolution des Workflows, nous introduisons une méthode pour évaluer ces systèmes d'opérations. Nous détaillons ici le concept de systèmes d'opérations. En effet, pour éviter de gérer les versions entre le modèle et les exécutions, nous avons préféré séparer les deux concepts en stratégie et incidences. En revanche, il est nécessaire de pouvoir suivre les évolutions des ces entités indépendantes. Car si les opérations locales d'une incidence n'influent pas sur son noyau, ses opérations globales sont notifiés à toute la génération à laquelle elle appartient. Par conséquent, il est nécessaire de disposer d'un moyen permettant d'évaluer les changements entre ces entités indépendantes. À un instant donné, deux systèmes d'opérations, appartenant ou non à une même génération, peuvent avoir intégrer différents opérations de changement. C'est alors qu'il est nécessaire de définir une notion de stabilité sur les systèmes d'opérations comme condition à leur évaluation. Se reporter à l'annexe C pour le détail des définitions : stabilité d'un système d'opérations du flot de contrôle, stabilité d'un système d'opérations du flot de contrôle à n-opérations près, proportionnalité des systèmes d'opérations purgés, proportionnalité des systèmes d'opérations du flot de contrôle, analogie des systèmes d'opérations du flot de contrôle et écart entre systèmes d'opérations. Détaillons maintenant le concept de « ***Noyau/Dérive(s)*** ».

Définition ***(Dérive minimale d'un système d'opérations)***. La dérive minimale, notée Δ^-, d'un système d'opération SO est définie par $\{i \in SO^{SO}_{Dérives}, k \in SO^{SO}_{Noyau} / \forall j \in (\{SO^{SO}_{Dérives}\} - \{i\}) \wedge card(\bowtie^i_k) \leq card(\bowtie^j_k))\}$.

Définition ***(Dérive maximale d'un système d'opérations)***. La dérive maximale, notée Δ^+, d'un système d'opération SO est définie par $\{i \in SO^{SO}_{Dérives}, k \in SO^{SO}_{Noyau} / \forall j \in (\{SO^{SO}_{Dérives}\} - \{i\}) \wedge card(\bowtie^i_k) \geq card(\bowtie^j_k))\}$. Ainsi $(\forall i \in SO^{SO}_{Dérives}, \exists k \in SO^{SO}_{Noyau})$ on a $(\{\Delta^-\} \propto^* \{i\} \propto^* \{\Delta^+\})$.

Après toutes ces définitions, il nous fallait trouver le moyen d'avoir une vision plus large sur l'évolution d'une génération de systèmes d'opérations. Après avoir étudier certains graphes existants (en étoile, de Kautz, de Bruijn, *etc.*), nous nous sommes rendu à l'évidence qu'il était impossible d'utiliser ces topologies, car l'alphabet constitué par les opérations de changement n'est pas fermé. Indubitablement, même si l'ensemble des opérations de changement est minimal et complet, l'alphabet constitué par les activités ne peut être restreint, puisque l'ensemble des tâches dans un modèle de procédé est non borné. De ce fait, il nous fallait trouver une topologie de graphe plus appropriée qui puisse saisir à la fois le concept *Stratégie/Incidence(s)* (et donc, l'idée de système d'opérations) et le paradigme *Noyau/Dérive(s)*. C'est dans ce contexte que nous faisons la définition suivante :

Définition ***(Graphe des arrangements lexicographique d'opérations centré sur le noyau avec attraction forte : G.A.L.O.C.N.A.FO)***. Un tel graphe valué et orienté est définit par un quadruplet $\mathbb{K}^\Delta = (N, R, V, K)$ avec $(\Delta = [op_{i=1...n}]$ et $\Delta_i = op_i)$ un n-uplet d'opérations étiquetées et ordonnées lexicographiquement où,

- N est un ensemble fini de nœuds. Ces sommets N_i représentent, individuellement, un ensemble d'opérations $N_i = [op_l, ..., op_k]$ étiquetées arrangées et ordonnées selon un ordre lexicographique défini par leurs estampilles, tel que $(card(N_i) \leq (card(\Delta) - 1)$;
- Une arrête $(N_{source}, N_{destination}) \in R$ si et seulement si :
  - le nœud N_{source} est un préfixe du nœud $N_{destination}$, c'est-à-dire, $\exists op_s \in N_{destination} / \ N_{source} = [op_l, ..., op_k]$ et $N_{destination} = [op_l, ..., op_k, op_s]$;
  - et $card(N_{destination}) = card(N_{source}) + 1$.
- Chaque arrête $(N_{source}, N_{destination})$ est valuée par $V(N_{source}, N_{destination}) = \{N_{destination}\} - \{N_{source}\} = op_s$;
- K est le nœud-noyau du graphe défini par l'ensemble des $(card(\Delta)!)$ arrangements possibles des Δ opérations dans $card(\Delta)$.

De même nous introduisons en annexe C les propriétés : Δ-rayon d'un $\mathbb{K}^\Delta$, de cardinalité, d'excentricité et de degré d'un nœud de ce graphe. Un $\mathbb{K}^\Delta$, dont l'ordre est donnée en annexe C, est planaire graphe. Toutes ces propriétés sont illustrées dans la figure C.4 du même annexe.

Définition ***(Graphe de suivi de l'évolution basé sur un*** $\mathbb{K}^\Delta$***)***. Un tel graphe, noté $\circledcirc^\Delta$, est graphe permettant de suivre les évolutions des incidences du flot de contrôle vis-à-vis de leur noyau. Δ représente l'ensemble des opérations de changement d'une génération. Ainsi, une stratégie stable S est représenté par le nœud-noyau du $\mathbb{K}^\Delta$ et les incidences- qui lui son proportionnelles -par leurs positions sur les nœuds du graphe (selon les opérations de changement qu'elles ont pu intégrer). La dérive maximale est atteinte par une incidence lorsqu'elle se trouve sur un nœud d'excentricité $(R_\Delta - 1)$, et la dérive minimale lorsqu'il existe une incidence se trouvant sur le noyau. L'incidence est alors analogue à la stratégie. Par extention aux travaux de *S. Rinderle et al.* [40], les dérives d'un systèmes d'opérations peuvent êtres ordonnées à travers les opérations de changement qu'elles ont pu intégrer selon leurs positions sur des nœuds d'excentricé croissante. L'excentricité croit partant du noyau du graphe vers les nœuds de cardinalité unitaire.

Selon les travaux présents dans la littérature (voir section 2.2), la propagation du changement vers des instances de workflow en exécution ou leur modification soulève le problème du contrôle et de la gestion de l'impact de ces changements sur les instances afin de prévenir des (états) d'exécutions incorrectes.

Généralement, les changements incontrôlés doivent être prohibés et des remaniements/compensations peuvent êtres nécessaires pour rétablir la cohérence de l'état d'exécution.

Dans le paradigme *Schéma/Instance(s)*, la cohérence d'une instance dépend de la correction dynamique de son schéma et de son état d'exécution actuel. D'une part, la correction dynamique du schéma garantit que chaque exécution partant de l'état initial est correcte. Habituellement, des outils d'analyse et de simulation sont employés par les analystes et les WfA pour vérifier cette propriété. D'autre part, une instance de Workflow est dans un état d'exécution cohérent si son état d'exécution actuel est correct conformément au nouveau schéma (après occurrence du changement). Parmi les propriétés de correction existantes (voir section 1.1.5), celle de *F. Casati et al.* [9] est la plus intéressante, car elle requiert que la totalité de l'historique d'exécution d'une instance (selon son ancien schéma) soit identique (on parle alors de légalité des primitives de changement) à une exécution selon le nouveau schéma. Quand cette propriété est satisfaite, l'instance peut être migrée vers le nouveau schéma sans aucun remaniement. Néanmoins, il est parfois nécessaire que certaines activités puissent êtres compensées (dans la limite du possible) pour que l'instance soit cohérente, ou que des variantes supplémentaires de Workflow doivent êtres introduits auxquelles l'instance puisse être conforme, et ainsi éviter la compensation des activités.

Dans ce contexte, s'il est primordial que notre *framework* puisse supporter la compensation, et par conséquent, définissons la topologie de graphe suivante :

Définition ***(Graphe des arrangements lexicographique d'opérations centré sur le noyau avec attraction faible : G.A.L.O.C.N.A.FA)***. Un tel graphe valué et orienté est définit par un quadruplet $\mathbb{I}^{\Delta} = (N, R, V, K)$ est un $\mathbb{K}^{\Delta}$ où,

- Si le nœud N_{source} est un préfixe du nœud $N_{destination}$, en plus, l'arrête $(N_{destination}, N_{source}) \in R$;
- et chaque arrête $(N_{destination}, N_{source})$ est valuée par $V(N_{destination}, N_{source}) = (\{N_{destination}\} - \{N_{source}\})^{-1} = {op_s}^{-1}$, c'est-à-dire, l'inverse de $V(N_{source}, N_{destination})$ pour $N_{source} = [op_l, ..., op_k]$ et $N_{destination} = [op_l, ..., op_k, op_s]$.

Les figures A.6, A.8, A.10 et A.12 illustrent la construction d'un $\mathbb{I}^{\Delta}$ avec un $\Delta - rayon$ allant de 1 à 4. La définition de cette topologie nous permet de suivre l'évolution réelle d'une génération de systèmes d'opérations selon un modèle avec compensation. Le degré d'un nœud d'un $\mathbb{I}^{\Delta}$ est donnée en annexe C.

Définition ***(Graphe de suivi de l'évolution basé sur un*** $\mathbb{I}^{\Delta}$***)***. Ce graphe, noté $\circledast^{\Delta}$, est un $\circledcirc^{\Delta}$ dont les incidences peuvent compenser des opérations de changement. La figure C.1 illustre un $\circledast^{[a,b,c]}$ dont le noyau représente une stratégie S ayant intégré les opérations de changement $\{a, b, c\}$ et ses dérives, les incidences $\{I_1^S, I_2^S, I_3^S, I_4^S\}$. $\{I_1^S\}$ est une dérive minimale, et $\{I_3^S, I_4^S\}$ sont des dérives maximales.

Pour que la définition de ces graphes ne soit pas seulement théorique, nous montrons qu'il est possible de linéariser ces graphes selon la propriété suivante :

Propriété ***(Linéarisation d'un*** $\mathbb{K}^{\Delta}$ ***ou d'un*** $\mathbb{I}^{\Delta}$***)***. La représentation (voir figures C.2 et C.3) d'un $\mathbb{K}^{\Delta}$ ou d'un $\mathbb{I}^{\Delta}$ peut être facilement linéarisée dans une structure linéaire par l'algorithme du tableau B.10 de l'annexe B.

Dans le cas d'une génération composée de stratégie et d'incidences, toute opération de changement globale se produisant au niveau d'une incidence, et qui représente un changement *ad-hoc* de l'exécution du procédé, est notifiée à la stratégie. Ainsi, cette dernière contient toutes les opérations de changement notifiées au sein de cette génération. Cependant, les changements au niveau de la stratégie, qui représentent l'évolution du modèle de Workflow, ne sont pas nécessairement intégrés par les incidences (du fait du critère de correction dynamique). Il est évident que chaque incidence reflète différemment les modifications de son noyau, en plus de ses changements locaux. C'est grâce à la topologie de graphe des arrangements lexicographique d'opérations centré sur le noyau qu'il possible de suivre l'évolution de changement dans une génération de systèmes d'opérations. Le noyau du graphe représente les opérations du changement global relatif à une génération, intégrées par une stratégie. Les différentes incidences sont positionnées sur les nœuds du graphe qui reflète les opérations de changement et l'ordre dans lequel ces

dernières ont pu intégrer. L'occurrence d'une opération de changement global dans une génération, aussi bien au niveau du noyau que des dérives, implique l'expansion du graphe. Les dérives qui ont intégré l'opération changent de position (c'est-à-dire, leur excentricité par rapport à leur noyau), et celles qui rejettent l'opération gardent la même position. Comme le graphe est linéarisable, son expansion ne nécessite pas réévaluer toutes les incidences par rapport à leur noyau pour connaître leur stade d'évolution vis-à-vis de la stratégie. De plus, l'algorithme de linéarisation permet soit de construire un graphe de suivi global pour toute la génération, soit, qu'au niveau de chaque système d'opérations, on puisse construire un graphe local pour l'évaluer par rapport à son noyau.

3.6 Étude de la complexité

Pour l'étude de la complexité de notre approche, nous nous intéressons seulement à l'évaluation du critère de correction, car la complexité des algorithmes des opérations est secondaire. Il est évident que notre critère de correction dynamique a une complexité inférieure à celles des approches basées sur l'évaluation des isomorphisme entre graphe, puisque ce problème est considéré être de classe NP. Même si par restriction, l'évaluation s'effectue sur des graphes de schémas de procédé ayant un étiquetage unique, et par une comparaison des matrices d'adjacence des deux schéma, sa complexité reste de l'ordre $O(n^2)$ où n est le nombre de nœuds (et par conséquent, le nombre des activités) [24]. Le fait que notre critère soit local à un système d'opérations, il en résulte que son évaluation est d'une complexité en $O(p * n)$ où n est le nombre des activités du modèle de Workflow, et p le nombre d'opérations de cycle de vie définies sur une activité.

3.7 Conclusion

Suite à nos réalisations, concluant ce chapitre par trois déductions finales. Premièrement, lorsqu'on veut accroître la flexibilité d'un WfMS par une gestion des modifications dynamiques des Workflows, nous sommes convaincu qu'il faut bien définir un langage de modélisation de haut niveau d'abstraction permettant une édition collaborative des procédés métiers. C'est avec cette idée que nous avons conçu notre approche. Deuxièmement, il est nécessaire de disposer de mécanismes limitant les changements possibles pour que les Workflows puissent rester (sémantiquement) valides vis-à-vis des règles générales qui régissent leurs métiers. Cette restriction doit aussi permettre de n'autoriser que les acteurs légitimes d'effectuer différents types de changements relativement à leur positionnement dans la hiérarchie du modèle organisationnel de l'organisation dans laquelle est déployé le WfMS. Finalement, le changement doit être intégré ou réfuté le plus rapidement possible pour une plus grande réactivité des analystes. D'où le choix d'un critère de correction tel que le notre. Illustrons maintenant l'innovation qu'apporte la dérive des systèmes d'opérations :

- Une incidence dérive d'une stratégie : Cette hiérarchie représente le cas de figure classique d'une instanciation d'un modèle de Workflow ;
- Une incidence dérive d'une autre incidence : Ce cas de figure se présente lorsqu'il est nécessaire d'exécuter un Workflow selon le modèle spécifique d'un Workflow ayant intégré des changements spécifiques qui n'ont pas été répertoriés au niveau de la stratégie ;
- Une stratégie dérive d'une incidence : Afin de ne pas avoir à modéliser un nouveau procédé à partir d'une feuille blanche, il est possible de définir un modèle de procédé à partir des spécificités d'une incidence ;
- Une stratégie dérive d'une autre stratégie : Dans les procédés dits « créatifs » , le besoin de réutiliser des modèles antérieurs est primordial. Ainsi, dans ce cas de figure, il est possible d'utiliser tout ou une partie d'une stratégie antérieure pour en créer une nouvelle.

« *Quand il est nécessaire de changer, il est nécessaire de ne pas changer.* » [Lucius Cary]

Conclusion et perspectives

Cette conclusion finale préfigure la fin de notre travail de master de recherche, il est donc nécessaire, qu'à ce stade du mémoire, nous puissions évaluer les résultats de notre réalisation dans son ensemble, et mettre en exergue les avantages, les limites et les perspectives ouvertes par l'approche que nous proposons.

Grâce à une étude de la flexibilité dans les systèmes de gestion des Workflows, nous avons pu contribuer à définir une approche innovante pour résoudre les problèmes de changements dynamiques dans ce genre de systèmes. L'introduction du paradigme de système d'opérations nous a permis d'estomper la distinction entre les notions de schéma et d'instance(s) dans la gestion des procédés métiers. Ainsi, un WfMS se voit gérer l'évolution d'entités, à un niveau plus haut, qui sont le noyau et les dérives. Le changement étant défini par des opérations sur un modèle de Workflow, ce qui accroît son édition collaborative aussi bien au niveau de la stratégie (représentant la vision stratégique du procédé), qu'au niveau des incidences (qui constituent une vision opérationnelle du procédé). Bien que notre intérêt porte sur des changements du flot de contrôle dans un modèle de Workflow, nous avons montré (en section 1.4) comment la définition de *framework*, que nous proposons, est-elle générique à l'égard de tout langage de modélisation de Workflow, de tout type de changement et surtout de toute politique de changement (voir section 1.1). Le concept de génération de systèmes d'opérations et de notification permet d'avoir une meilleure traçabilité du changement. Les opérations d'exportation et de publication permettent une réutilisation du changement, puisque celui-ci est décrit par un ensemble ordonné et cohérent d'opérations permettant de créer un nouveau système d'opérations (utilisant un sous-ensemble d'opérations ou l'ensemble entiers des opérations globales de changement). Pour une gestion du changement plus efficiente, nous avons proposé une topologie de graphe pour le suivi de l'évolution des systèmes d'opérations. Nous avons montré que la structure de cette topologie peut être linéarisée par un algorithme récursif, ne nécessitant pas de maintenir le graphe et permettant de positionner, chaque système d'opérations d'une génération, sur le nœud qui correspond aux opérations de changement que ce derniers a pu intégrer. De ce fait, lorsqu'il s'agit d'opérations de changements concurrents, il est alors possible d'ordonnancer les incidences par leur excentricité par rapport à leur noyau.

Bien sûr, tout modèle présente des limites, sans lesquelles il n'est point possible d'affirmer ses contributions. À ce stade d'avancement du projet, il nous ai nécessaire de réfléchir sur les points suivants. Comment peut-on utiliser des techniques de preuves avancées (*Automated theorem proving*, par exemple) pour évaluer notre critère de correction par rapport aux problèmes de changement, cités en section 1.1.5. En particulier, serait-il efficace en présence de boucle dans le flot de contrôle. Comme le critère de correction est défini sur l'historique purgé des opérations, ne pourrait-on pas factoriser certaines d'entre elles sans pour autant risquer de perdre du contrôle sur l'évolution et avoir une érosion sur la structure du modèle de Workflow, ce qui conduira à réduire la complexité.

Incontestablement, un projet de cette nature, qui plus est de recherche, ne se termine jamais. Mais le prototypage serait une étape importante de son application réel dans un WfMS. Puisque, dans ce mémoire, nous présentons déjà les algorithmes des opérations (voir annexe B), ainsi que le modèle structurel de notre *framework* (figures et de l' annexe A), la perspective première est de s'intéresser aux aspects de son implémentation. Grâce au haut niveau de nos spécifications nous pourrons intégré notre *framework* en tant que *plug-in* à un système de gestion de Workflow réel, comme celui développé par l'équipe ECOO. L'implémentation impliquera sans doute la problématique supplémentaire de l'ordonnancement

des opérations de changements concurrents.

Ceci dit, et à notre connaissance, il n'existe point, dans la littérature, une contribution voulant résoudre le problème de l'évolution des Workflows, selon la même vision que nous présentons dans ce mémoire. C'est ainsi, que nous pouvons affirmer, sans prétention aucune, l'innovation que constitue notre proposition. Cependant, voici les perspectives ouvertes par notre approche.

Le paradigme de système d'opérations faisant en sorte que l'exécution dans un WfMS revient à une observation d'une suite cohérente d'opérations, il maintenant possible de réfléchir à la distribution des moteurs d'exécution et à la problématique de la confidentialité introduite par cette distribution. Autrement dit, si une organisation décide de déléguer une partie de son procédé, quel est la politique qu'il est nécessaire d'introduire pour que l'externalisation ne provoque un risque de divulgation du cœur de métier.

L'intérêt que nous portons pour la correction structurelle et comportementale du changement (voir section 1.3) implique de s'intéresser autant à sa correction sémantique, à l'intention même du changement et au contrôle d'accès au changement. D'où la question de la légalité et de l'éthique de la conduite des changements dans un procédé métier.

Dans certain processus, la politique du changement voudrait qu'il soit possible et même nécessaire de recourir à une compensation des activités pour annuler les travaux réalisés pour garantir une exécution sémantiquement correcte. Selon notre approche faisant que le changement est conduit par des opérations, il est intéressant de s'intéresser au fait de définir la compensation en terme d'opérations (dans la limite de leur commutativité, voir annexe E). Cette problématique de la commutativité du changement nous amènera, par l'utilisation du graphe de suivi de l'évolution basé sur un $\mathbb{I}^{\Delta}$ (défini en section 3.2), à un inspecter, par exemple, dans quelle limite l'insertion d'une activité pourrait nécessiter une compensation pour qu'elle soit correcte. Comme l'histoire de l'exécution du procédé est contenue dans les systèmes d'opérations, n'est-il pas possible, pour une compensation, de remettre en cause l'historique du changement et de l'exécution (ce qui est prohibé par toutes les approches existantes) puisqu'il est toujours possible de les ré-exécuter.

Dans le contexte de l'organisation scientifique du travail, et en particuliers le toyotisme qui, situé dans la lignée du taylorisme et du fordisme, repose sur trois principes : un enrichissement des tâches, une plus grande participation des salariés, une mise en place de cercles de qualité où ouvriers et ingénieurs se réunissent volontairement pour améliorer la qualité et la productivité du travail. En rendant les tâches plus flexibles et en associant davantage les salariés au développement de l'entreprise, le toyotisme s'impose comme amélioration du taylorisme et du fordisme. Dans cet ordre d'idées et selon notre approche, la publication des opérations des incidences vers la stratégie permet d'améliorer les échanges entre niveau opérationnel (c'est-à-dire, l'incidence) et niveau décisionnel (c'est-à-dire, la stratégie). Si l'on considère que la notion de stratégie (en sous entendant le modèle du Workflow) est la capitalisation du savoir-faire des ingénieurs, elle peut être décrite par « que faut-il faire et comment le faire ». Alors qu'une incidence (sous entendu l'exécution du modèle) est une matérialisation de l'expérience des ouvriers (quand il s'agit de dévier du modèle par des changements *ad-hoc*) reflétant l'exécution du procédé selon les particularités de l'environnement réel, et est décrit par l'affirmation « c'est ainsi que nous le faisons ». Par une exploitation avancée du graphe de suivi de l'évolution basé sur la topologie du graphe des arrangements lexicographique d'opérations centré sur le noyau, et en utilisant des techniques de fouille (*Mining*) sur le changement, il serait possible de déterminer des motifs de modification des incidences. Et ainsi, lors de l'exécution du procédé, il deviendrait possible de proposer des alternatives optimisées quand au choix d'effectuer des changements *ad-hoc*. C'est de ce dire si certaines incidences ont divergées de la stratégie, ne serait-il pas judicieux que lorsqu'il s'agit d'effectuer un changement dans une autre incidence de proposer des motifs de changement déjà présent. D'autre part, ces motifs de changement, une fois analyser, peuvent êtres publiés vers la stratégie, et donc améliorer le savoir-faire par l'expérience et le pragmatisme répendant à l'affirmation « Voilà réellement comment faut-il le faire ». Et enfin, il serait pas moins intéressent de construire une algèbre sur les systèmes opérations.

« *Regarder loin, c'est regarder tôt* » [Hubert Reeves] ... Et regarder tôt, c'est regarder juste.

Annexe A

Illustrations

Comme le montre la figure A.1, une activité d'une instance particulière du procédé peut nécessiter un accès à des ressources externes à l'outil de gestion de procédé (par exemple, des objets métiers, des documents, des bases de données, des bus de messages, *etc.*). Tandis que l'exécution est à la charge du moteur, la description et la maîtrise du comportement de ces activités est toutefois sous la responsabilité du programmeur. Néanmoins une question se pose : comment ces deux entités doivent-elles coopérer ?

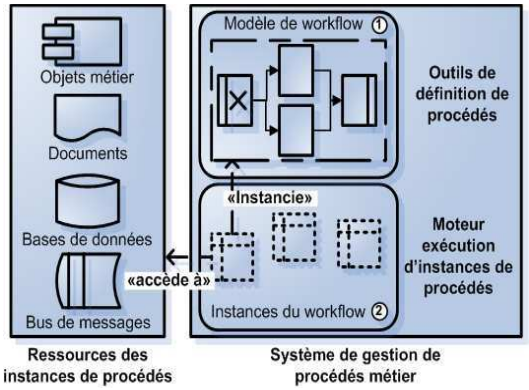


FIG. A.1 – Environnement de définition et d'exécution d'un procédé.

À ce stade, il est nécessaire de distinguer deux grandes catégories de produits et donc deux marchés ; celui du *Process Modeling* et celui du Workflow, même si certains éditeurs intègrent de plus en plus les deux dimensions. Des systèmes de gestion de procédés métiers tels que : *Staffware*, *IBM MQSeries*, *COSA*, *etc.*, permettent une modélisation générique (c'est-à-dire, indépendants du secteur d'activité des entreprises auxquelles ils sont destinés) et des capacités d'exécution pour des procédés structurés. En plus des systèmes de gestion de procédés purs (c'est-à-dire, qui sont strictement dédiés à la gestion de Workflow, par opposition à ceux qui ont intégrés cette dimension au cours de leur cycle de développement après l'apparition de nouveaux besoins de rationalisation du travail), bien d'autres systèmes ont adopté le paradigme de Workflow. Prenons comme exemple les prologiciels de planification de ressources d'entreprise (*Entreprise Ressource Planning* : ERP) comme ceux édités par *SAP*, *Oracle Applications*, *The Sage Group* ou encore *Microsoft Dynamics*, les logiciels de gestion de la relation client (*Customer Relationship Management* : CRM), les systèmes de gestion de la chaîne de production (*Supply Chain Management* : SCM), *etc.*. Remarquablement, tous ces systèmes incluent des représentations particulières de métamodèles cohérents de Workflow. En conséquence de leur succès, et vu la situation du marché des

ERP en Europe, Moyen-Orient et Afrique [51] (11,6 milliards de dollars en prévision de l'année 2011, source : *journaldunet.com*), leur réputation n'est plus à faire, puisqu'ils ont indéniablement démontré leur potentiel de modélisation et d'automatisation.

La figure suivante est un exemple de modèle d'un procédé de traitement des demandes d'immigration pour un service consulaire [11] exprimé selon le langage de modélisation défini en section 1.1.1.

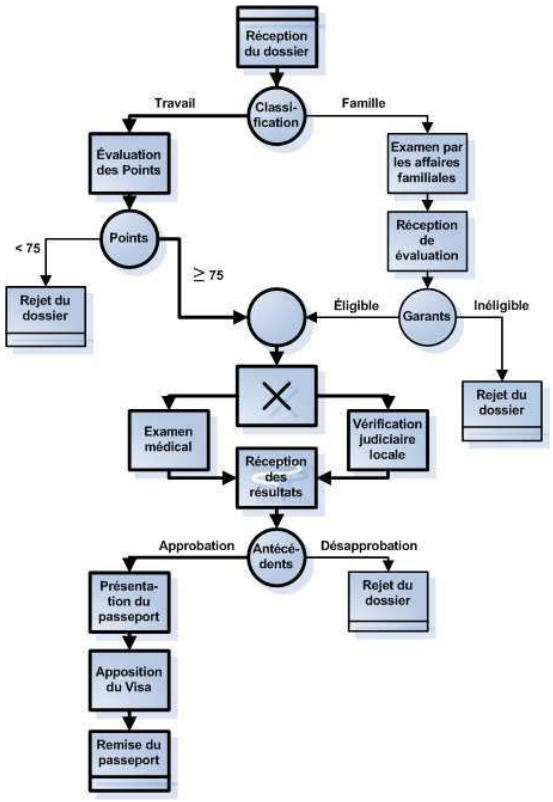


FIG. A.2 – Modèle d'un procédé de traitement des demandes d'immigration.

La figure A.3 illustre l'analogie entre l'évolution d'une stratégie S et la construction du schéma de flot de contrôle associé par application d'opérations estampillées aussi bien de changement, que de cycle de vie. Les opérations de changement S devant incontestablement êtres notifiées à tous les systèmes d'opérations dérivant de S, elles sont pour cela considérées comme globales. La création et l'activation d'une stratégie étant des opérations de cycle de vie, elles sont intégrées comme des opérations privées.

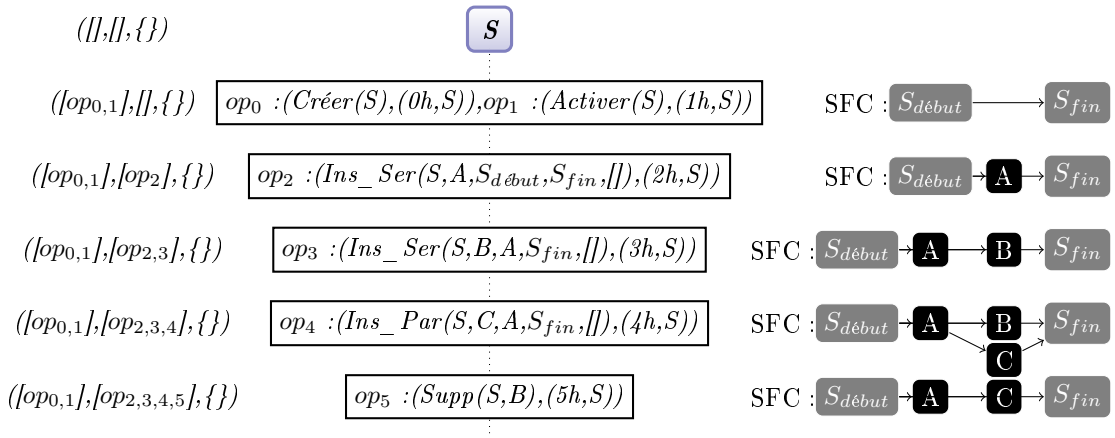


FIG. A.3 – Exemple d'évolution d'une stratégie S.

L'illustration A.4 montre une incidence I_1^S dérivant d'une stratégie S et une incidence $I_2^{I_1^S}$ qui dérive de I_1^S. La création des incidences se fait par exportation des opérations globales du système d'opérations dont elles dérivent. Etant donné que chaque système d'opération possède sa propre horloge logique, il ainsi possible d'ordonner les opérations au sein d'un même système, que les ordonner d'une manière globale. Ceci est illustré par les égalités entre les estampilles. Le mécanisme d'exportation permet de créer une incidence à partir d'une autre incidence. Cette technique innovante offre la possibilité créer un Workflow dont le modèle est issu du modèle d'une incidence ayant intégré des changements *ad-hoc* qui n'ont pas était notifiés à la stratégie dont elle dérive. Et d'autre part, de créer une stratégie (implicitement un modèle de Workflow) à partir d'une autre stratégie ou même d'une incidence.

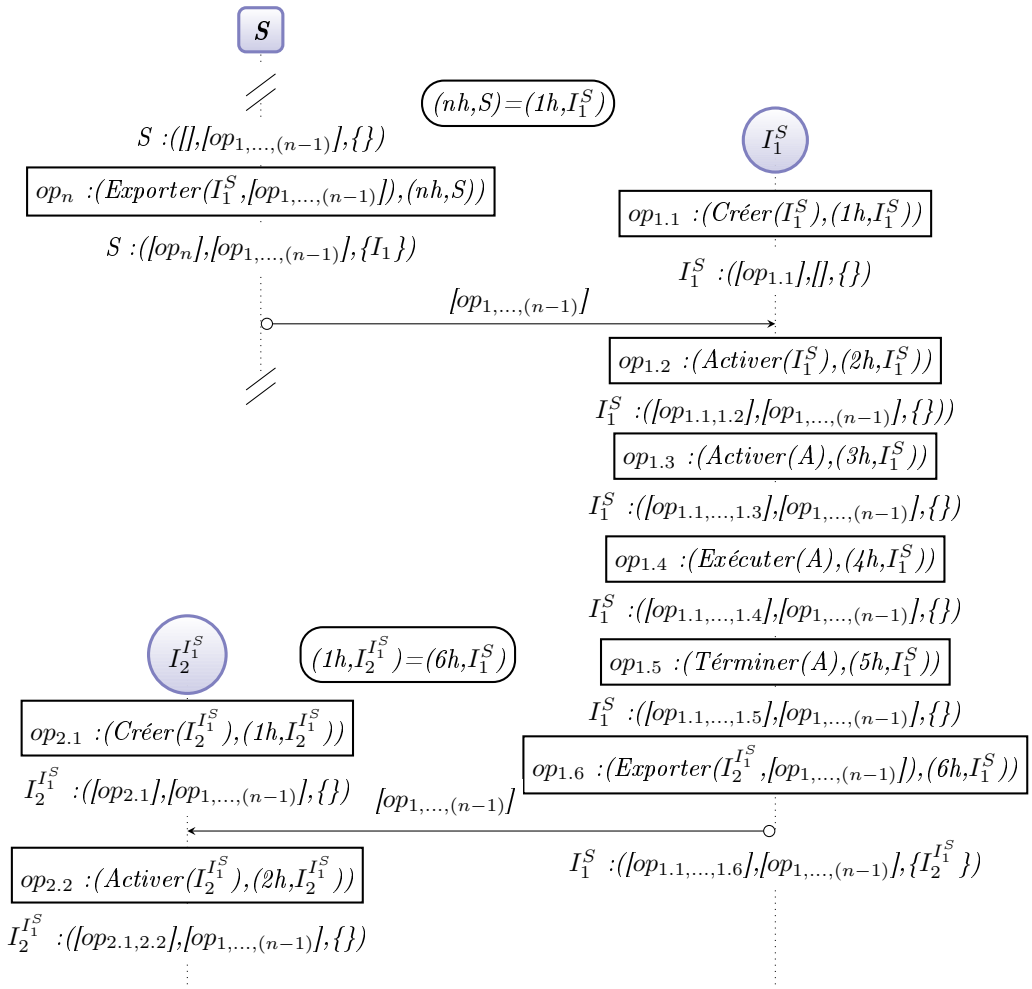


FIG. A.4 – Illustration de l'exportation des opérations de changements des systèmes d'opérations.

La figure A.5 illustre un exemple d'une incidence I_1^S dérivant d'une stratégie S, et comment la portée des opérations de changement conditionne leur notification. L'opération $op_{1.n}$ est privée à l'incidence I_1^S. Elle constitue une opération de changement *ad-hoc* qui n'est pas notifié au noyau S. Par contre, $op_{1.(n+1)}$ à une portée globale, c'est pour cela qu'elle doit être notifiée aussi bien au noyau qu'à toute les dérives (voir l'algorithme du tableau B.5). Ici, nous représentons seulement son noyau S, qui intègre cette opération $op_{1.(n+1)}$. La stratégie devrait ensuite notifier cette même opération à toute la génération des systèmes d'opérations à laquelle elle appartient. L'opération op_{n+2} est, par exemple, une opération de changement au niveau de la stratégie S, sa portée est donc globale. S notifie alors sa dérive I_1^S de l'occurrence de cette opération.

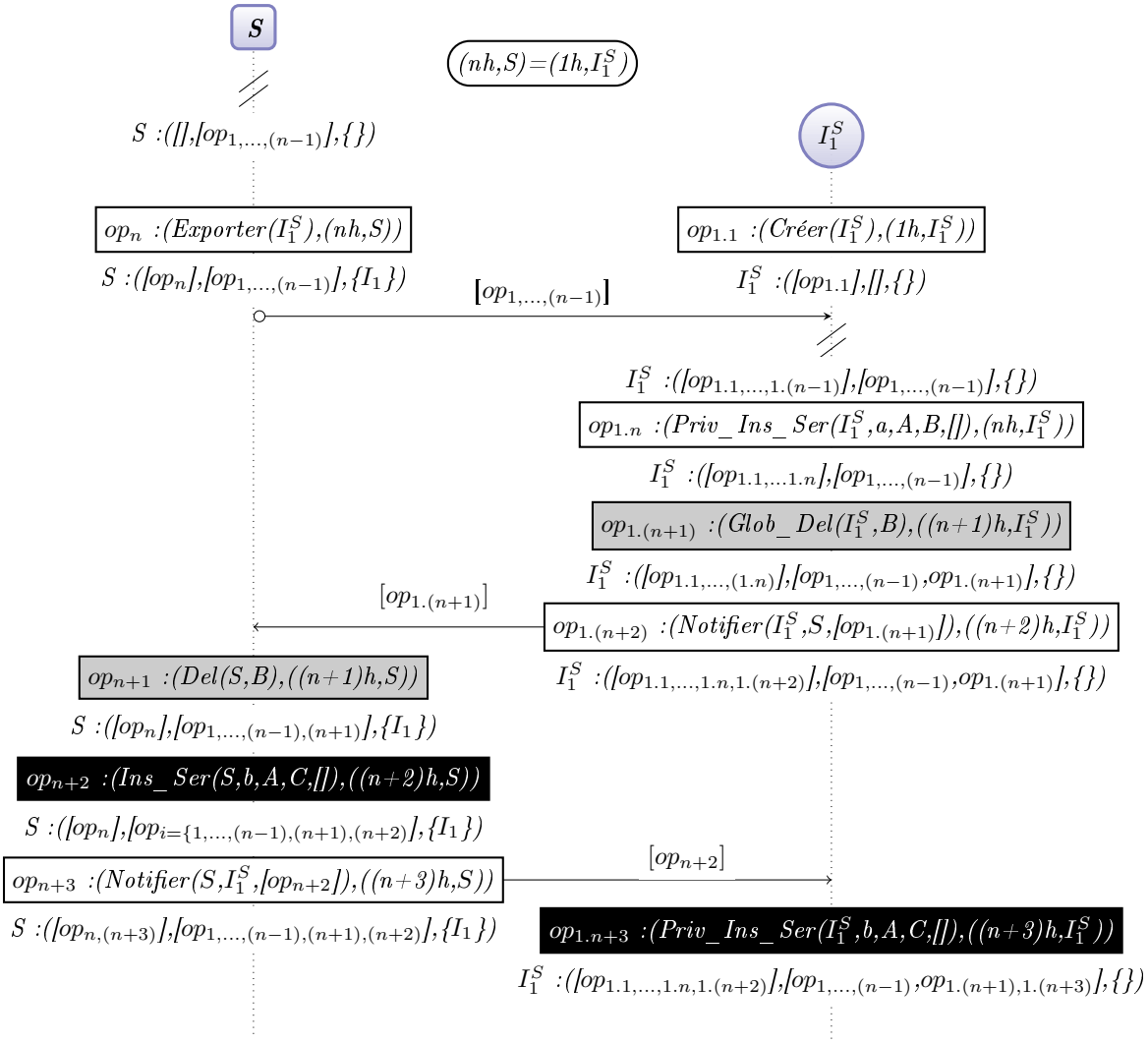


FIG. A.5 – Exemple de la dérive des systèmes d'opérations et de la projection des opérations.

FIG. A.6 – Graphe des arrangements lexicographique d'opérations (Δ=[a]) centré sur le noyau.

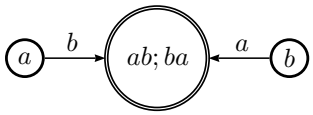


FIG. A.7 – *G.A.L.O.C.N.A.FO* avec (Δ=[a,b])

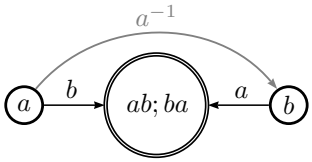


FIG. A.8 – *G.A.L.O.C.N.A.FA* avec (Δ=[a,b]).

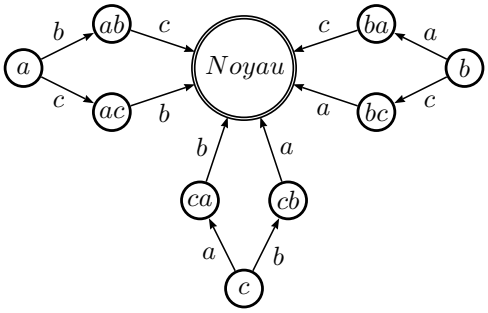


FIG. A.9 – *G.A.L.O.C.N.A.FO* avec (Δ=[a,b,c]).

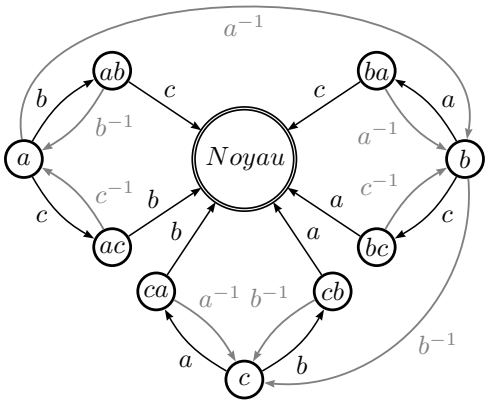


FIG. A.10 – *G.A.L.O.C.N.A.FA* avec (Δ=[a,b,c]).

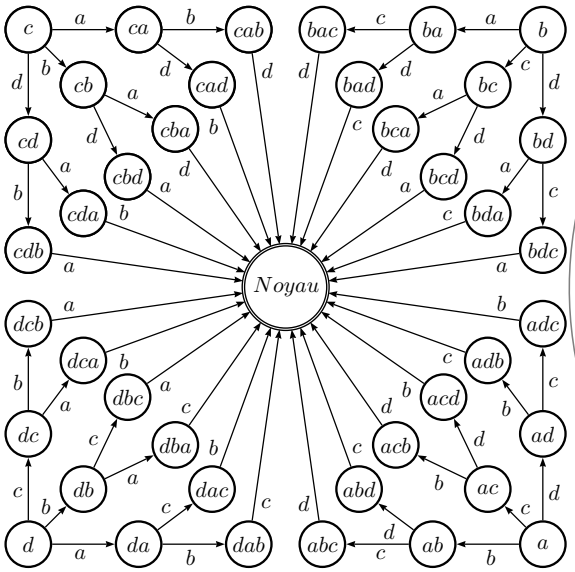


FIG. A.11 – *G.A.L.O.C.N.A.FO* avec (Δ=[a,b,c,d]).

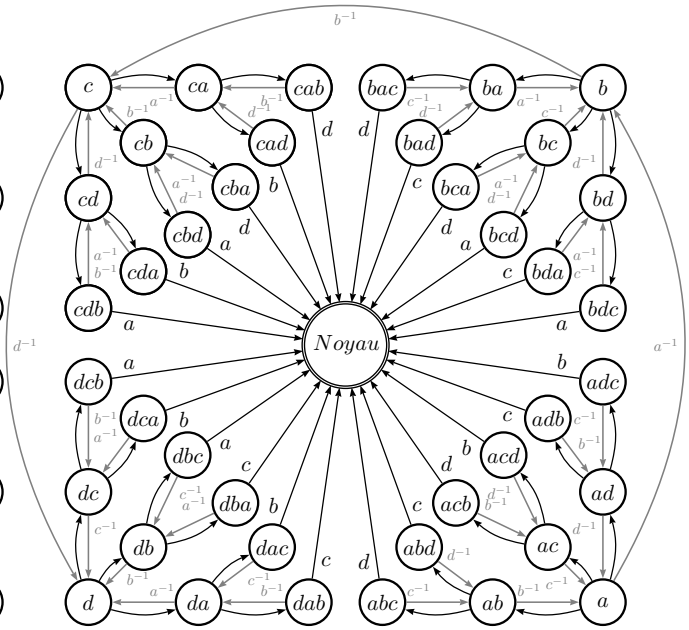


FIG. A.12 – *G.A.L.O.C.N.A.FA* avec (Δ=[a,b,c,d]).

Il faut savoir que ce modèle reste générique par rapport à tout langage de modélisation des Workflow et à tout type de changement concernant les flots de composants d'un procédé métier. Ici, nous exprimons seulement le flot de contrôle.

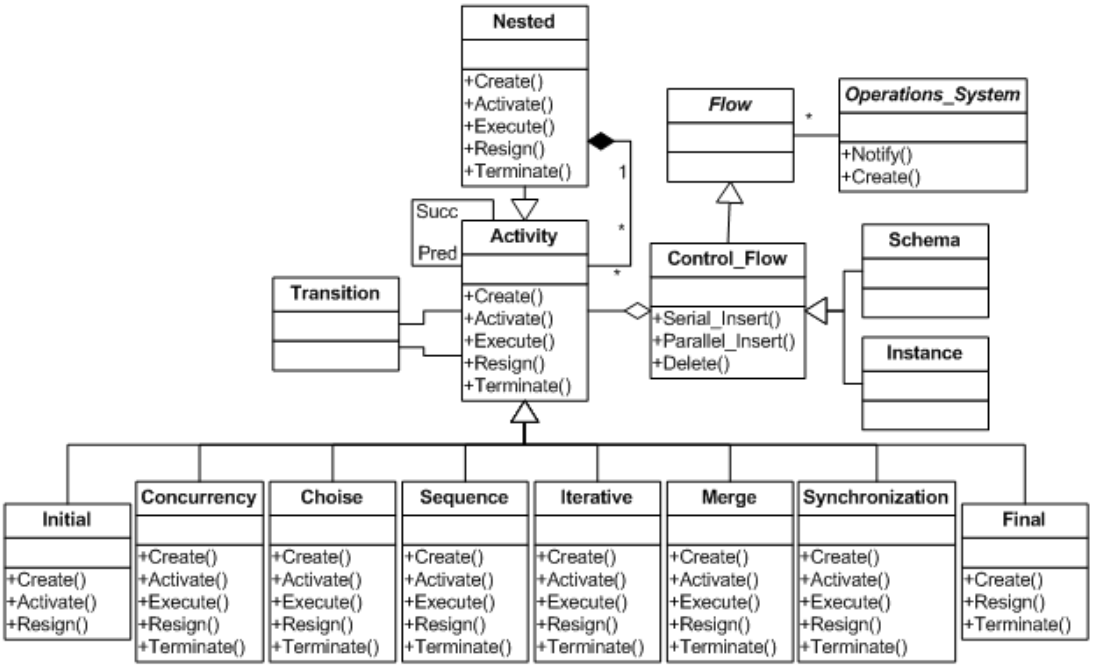


FIG. A.13 – Diagramme de classes des flots de « *Morphflow* ».

Le modèle de ce diagramme de classes qui présente la hiérarchie des opérations est construit selon les patrons de conception *Decorator*, *Template Method*, *Visitor* et *Strategy* [52].

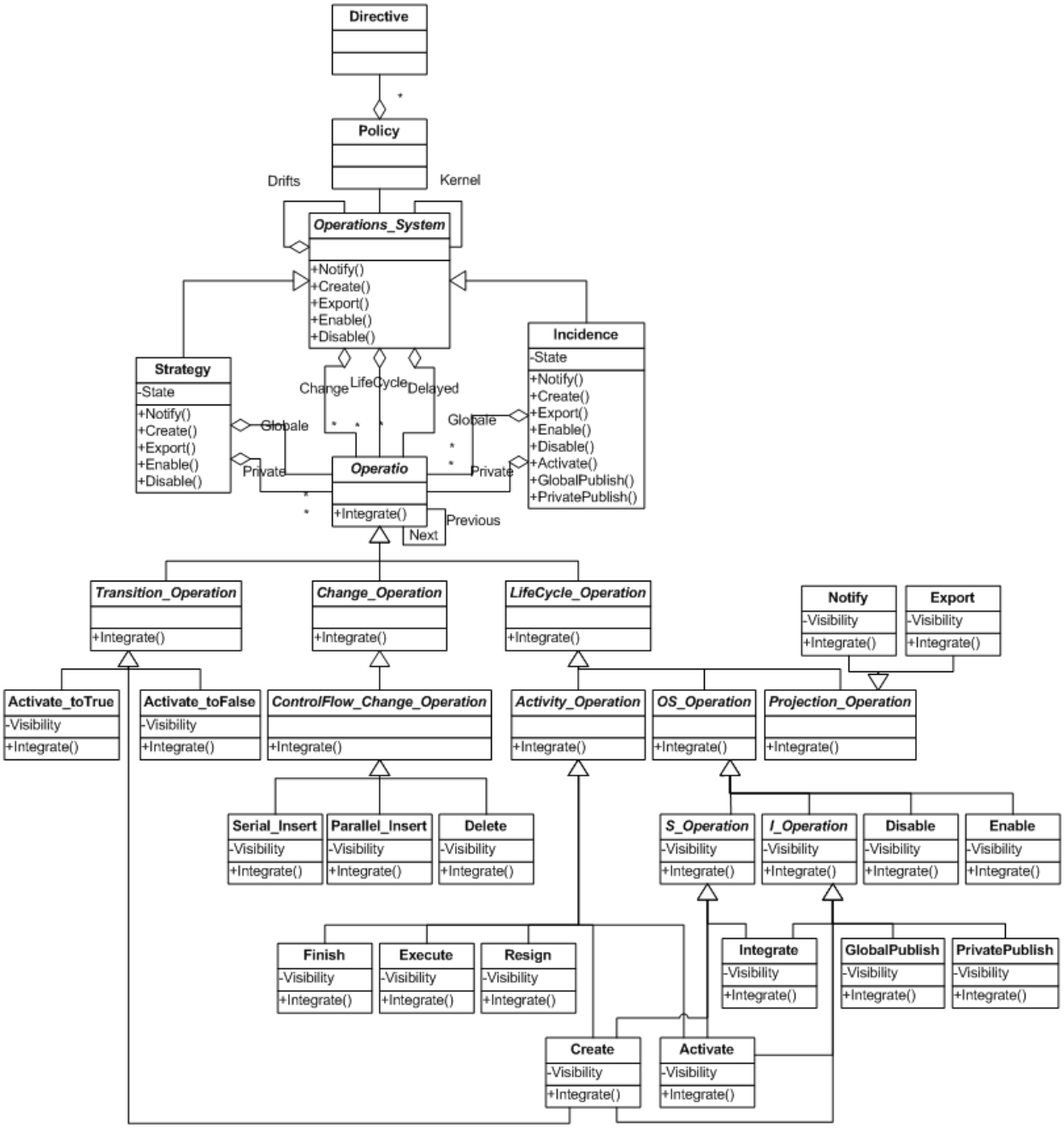


FIG. A.14 – Modèle structurel de « *Morphflow* ».

Annexe B

Algorithmes

Fonction $lop :: Créer($ so : **SO**, $[op_i]$: **ops**$)$: **bool**
résultat : **bool** ;
$Créer(so, a_{Initiale})$;
$Créer(so, a_{Finale})$;
Pour $\forall op_i \in [op_i]$ **faire**
Si (op_i *[Code effectif]*) **Alors**
résultat $\leftarrow$ **vrai** ;
[Selon la portée]
$OP^{so}_{Changement} \leftarrow OP^{so}_{Changement} + \{opi\}$;
Retourner Vrai ;
Sinon
résultat $\leftarrow$ **faux** ;
Fin Si
Fin Pour
Si (résultat) **Alors**
[Selon la portée]
$OP^{so}_{Cycledevie} \leftarrow OP^{so}_{Cycledevie} + \{lop\}$;
$Activer(so)$;
Retourner vrai ;
Sinon
Retourner faux ;
Fin Si
Fin

TAB. B.1 – Algorithme de création d'un système d'opérations.

Fonction $lop :: Créer, Activer, Abandonner, Exécuter, Terminer($ so : **SO**, a : **A**$)$: **bool**
  Si (lop *[Code effectif]*) **Alors**
    [Selon la portée]
    $OP^{so}_{Cycledevie} \leftarrow OP^{so}_{Cycledevie} + \{lop\}$;
    Retourner vrai ;
  Sinon
    Retourner faux ;
  Fin Si
Fin

TAB. B.2 – Algorithmes de cycle de vie d'une Incidence.

Fonction $lop :: Exporter($ s : **S**, $so_{nouveau}$: **SO**$)$: **bool**
  $[op_i]$: **ops** ;
  Si $(V^s = \{Exportable\})$ **Alors**
    Si $(P_{PCT} = \{Différé\})$ **Alors**
      $[op_i] \leftarrow OP^s_{Changement} + OP^S_{Différées}$;
    Sinon
      $[op_i] \leftarrow OP^s_{Changement}$;
    Fin Si
  Sinon
    $[op_i] \leftarrow \emptyset$;
  Fin Si
  $[\ Créer(so_{nouveau}, [op_i])]$
  $[\ SO^s_{Dérives} \leftarrow SO^s_{Dérives} + \{so_{nouveau}\}\ ]$
  $[\ SO^{so_{nouveau}}_{Noyau} \leftarrow \{s\}\ ]$
  Si $(Notifier(s, so_{nouveau}, [op_i]))$ **Alors**
    $OP^s_{Cycledevie} \leftarrow OP^s_{Cycledevie} + \{lop\}$;
    Retourner vrai ;
  Sinon
    Retourner faux ;
  Fin Si
Fin

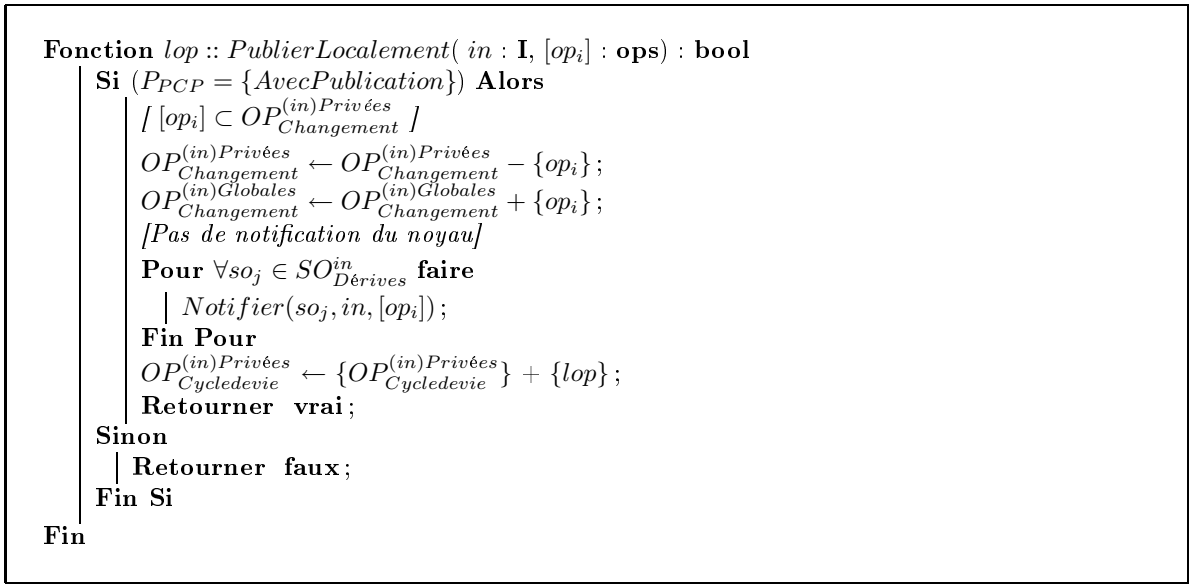
TAB. B.3 – Algorithme d'exportation des opérations de changement d'une Stratégie.

Fonction $lop :: Exporter(\ in : \mathbf{I},\ so_{nouveau} : \mathbf{SO}) : \mathbf{bool}$
  $[op_i] : \mathbf{ops}$;
  Selon que
    $V^{in} = \{GlobalesExportable\}$:
    [Les opérations de changement globales seulement]
    Si $(P_{PCT} = \{Diff\acute{e}r\acute{e}\})$ **Alors**
      $[op_i] \leftarrow OP^{(in)Globales}_{Changement} + OP^{S}_{Diff\acute{e}r\acute{e}es}$;
    Sinon
      $[op_i] \leftarrow OP^{(in)Globales}_{Changement}$;
    Fin Si
    $V^{in} = \{PriveesExportable\}$:
    [Les opérations de changement globales et privées]
    Si $(P_{PCT} = \{Diff\acute{e}r\acute{e}\})$ **Alors**
      $[op_i] \leftarrow OP^{in}_{Changement} + +OP^{S}_{Diff\acute{e}r\acute{e}es}$;
    Sinon
      $[op_i] \leftarrow OP^{in}_{Changement}$;
    Fin Si
    $V^{in} = \{NonExportable\}$:
    $[op_i] \leftarrow \emptyset$;
  Fin Selon que
  [$Cr\acute{e}er(so_{nouveau}, [op_i])$ *]*
  [$SO^{in}_{D\acute{e}rives} \leftarrow SO^{in}_{D\acute{e}rives} + \{so_{nouveau}\}$ *]*
  [$SO^{so_{nouveau}}_{Noyau} \leftarrow \{in\}$ *]*
  Si $(Notifier(in, so_{nouveau}, [op_i]))$ **Alors**
    $OP^{in}_{Cycledevie} \leftarrow OP^{in}_{Cycledevie} + \{lop\}$;
    Retourner vrai ;
  Sinon
    Retourner faux ;
  Fin Si
Fin

TAB. B.4 – Algorithme d'exportation des opérations de changement d'une Incidence.

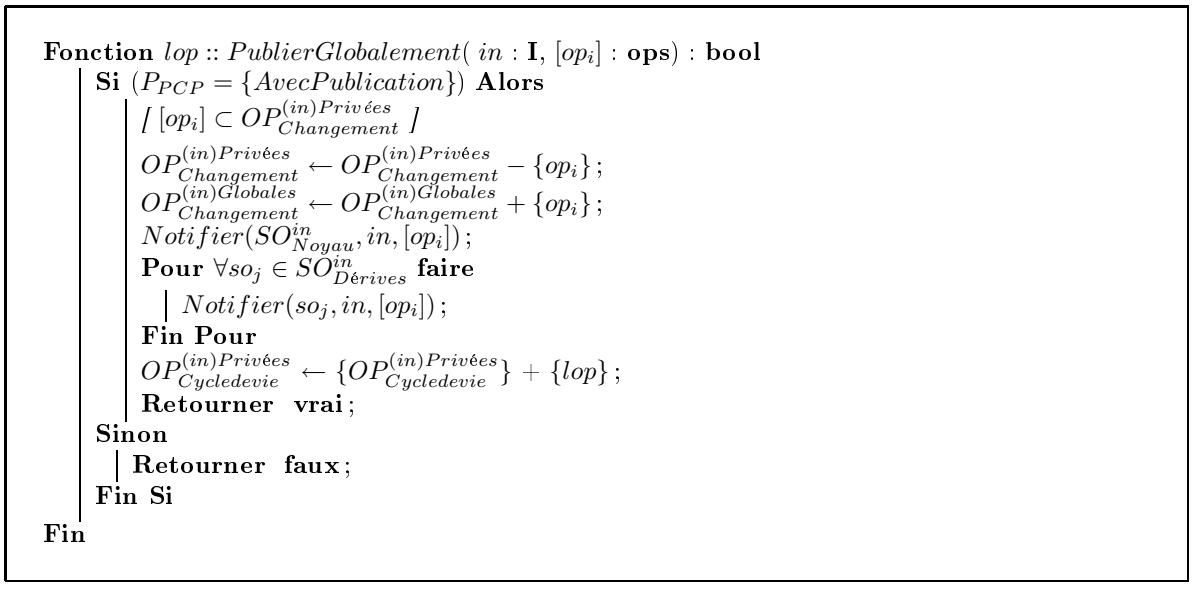
```
Fonction lop :: Notifier( so,so_expéditeur : SO, [op_i] : ops) : bool
    résultat : bool;
    Pour ∀op_i ∈ [op_i] faire
        Si (Intégrer(so, op_i) [Intégration locale] ) Alors
            Si (P^{op_i} = {Globale}) Alors
                résultat ← vrai;
                Pour ∀so_j ∈ ({SO^{so}} − {so_expéditeur}) faire
                    Si (Notifier(so_j, so, op_i)) Alors
                        résultat ← vrai;
                    Sinon
                        résultat ← faux;
                    Fin Si
                Fin Pour
            Fin Si
        Sinon
            résultat ← faux;
        Fin Si
    Fin Pour
    Si (résultat) Alors
        OP^{so}_{Cycledevie} ← OP^{so}_{Cycledevie} + {lop};
        Retourner vrai;
    Sinon
        Retourner faux;
    Fin Si
Fin
```

TAB. B.5 – Algorithme de notification des opérations de changement d'un système d'opérations.



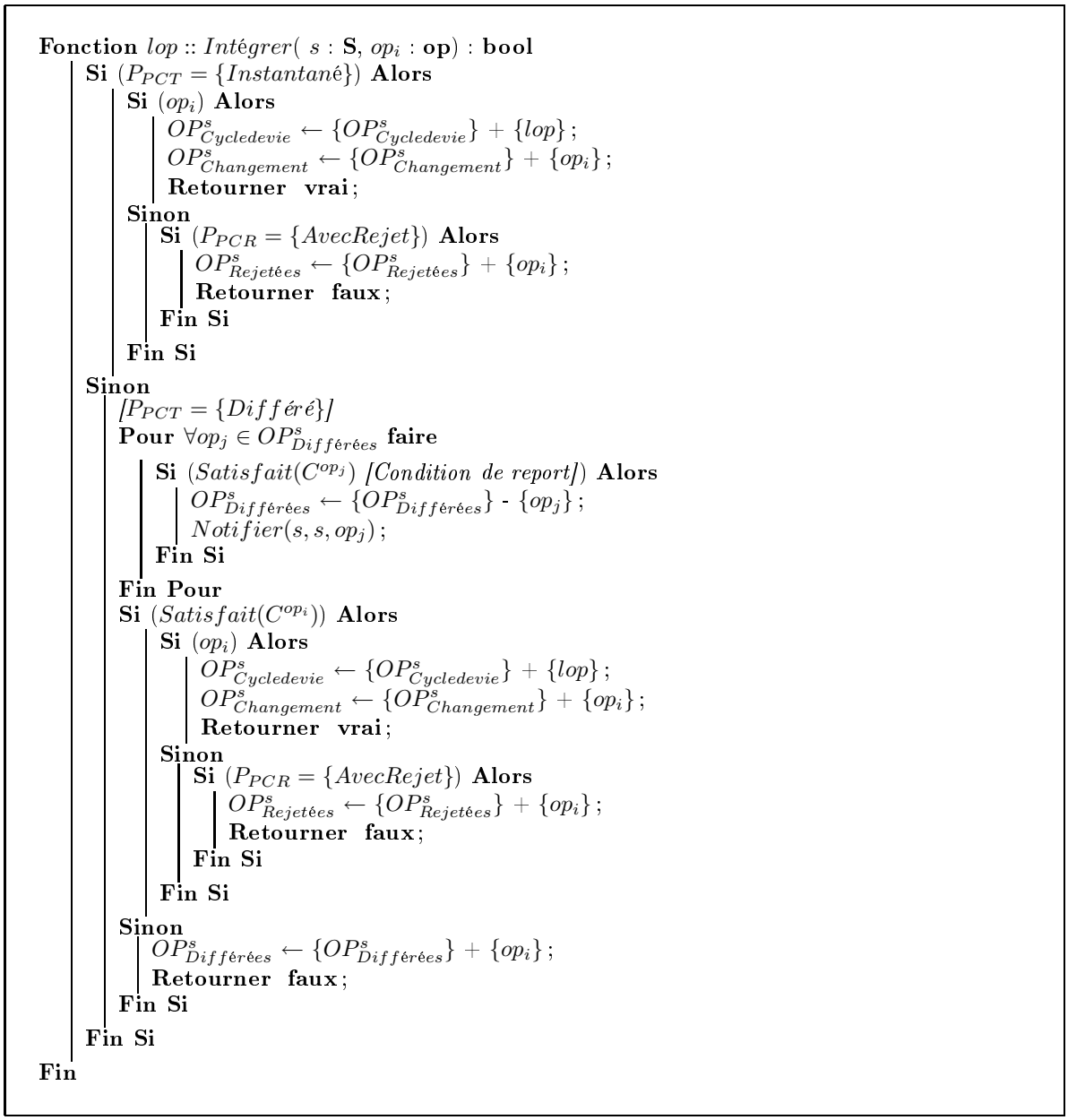
Fonction $lop :: PublierLocalement($ in : **I**, $[op_i]$: **ops**$)$: **bool**
Si $(P_{PCP} = \{AvecPublication\})$ **Alors**
$/\ [op_i] \subset OP^{(in)Privées}_{Changement}\ /$
$OP^{(in)Privées}_{Changement} \leftarrow OP^{(in)Privées}_{Changement} - \{op_i\}$;
$OP^{(in)Globales}_{Changement} \leftarrow OP^{(in)Globales}_{Changement} + \{op_i\}$;
[Pas de notification du noyau]
Pour $\forall so_j \in SO^{in}_{Dérives}$ **faire**
$Notifier(so_j, in, [op_i])$;
Fin Pour
$OP^{(in)Privées}_{Cycledevie} \leftarrow \{OP^{(in)Privées}_{Cycledevie}\} + \{lop\}$;
Retourner vrai ;
Sinon
Retourner faux ;
Fin Si
Fin

TAB. B.6 – Algorithme de publication locale des opérations de changement d'une Incidence.



Fonction $lop :: PublierGlobalement($ in : **I**, $[op_i]$: **ops**$)$: **bool**
Si $(P_{PCP} = \{AvecPublication\})$ **Alors**
$/\ [op_i] \subset OP^{(in)Privées}_{Changement}\ /$
$OP^{(in)Privées}_{Changement} \leftarrow OP^{(in)Privées}_{Changement} - \{op_i\}$;
$OP^{(in)Globales}_{Changement} \leftarrow OP^{(in)Globales}_{Changement} + \{op_i\}$;
$Notifier(SO^{in}_{Noyau}, in, [op_i])$;
Pour $\forall so_j \in SO^{in}_{Dérives}$ **faire**
$Notifier(so_j, in, [op_i])$;
Fin Pour
$OP^{(in)Privées}_{Cycledevie} \leftarrow \{OP^{(in)Privées}_{Cycledevie}\} + \{lop\}$;
Retourner vrai ;
Sinon
Retourner faux ;
Fin Si
Fin

TAB. B.7 – Algorithme de publication globale des opérations de changement d'une Incidence.



```
Fonction lop :: Intégrer( s : S, op_i : op) : bool
    Si (P_PCT = {Instantané}) Alors
        Si (op_i) Alors
            OP^s_Cycledevie ← {OP^s_Cycledevie} + {lop};
            OP^s_Changement ← {OP^s_Changement} + {op_i};
            Retourner vrai;
        Sinon
            Si (P_PCR = {AvecRejet}) Alors
                OP^s_Rejetées ← {OP^s_Rejetées} + {op_i};
                Retourner faux;
            Fin Si
        Fin Si
    Sinon
        [P_PCT = {Différé}]
        Pour ∀op_j ∈ OP^s_Différées faire
            Si (Satisfait(C^{op_j}) [Condition de report]) Alors
                OP^s_Différées ← {OP^s_Différées} - {op_j};
                Notifier(s, s, op_j);
            Fin Si
        Fin Pour
        Si (Satisfait(C^{op_i})) Alors
            Si (op_i) Alors
                OP^s_Cycledevie ← {OP^s_Cycledevie} + {lop};
                OP^s_Changement ← {OP^s_Changement} + {op_i};
                Retourner vrai;
            Sinon
                Si (P_PCR = {AvecRejet}) Alors
                    OP^s_Rejetées ← {OP^s_Rejetées} + {op_i};
                    Retourner faux;
                Fin Si
            Fin Si
        Sinon
            OP^s_Différées ← {OP^s_Différées} + {op_i};
            Retourner faux;
        Fin Si
    Fin Si
Fin
```

TAB. B.8 – Algorithme d'intégration d'une opération de changement dans une Stratégie.

```
Fonction lop :: Intégrer( in : I, op_i : op) : bool
    Si (P_PCT = {Instantané}) Alors
        Si (op_i) Alors
            Si (P^{op_i} = {Globale}) Alors
                OP^{(in)Privées}_{Cycledevie} ← {OP^{(in)Privées}_{Cycledevie}} + {lop};
                OP^{(in)Globales}_{Changement} ← {OP^{(in)Globales}_{Changement}} + {op_i};
                Retourner vrai;
            Sinon
                OP^{(in)Privées}_{Cycledevie} ← {OP^{(in)Privées}_{Cycledevie}} + {lop};
                OP^{(in)Privées}_{Changement} ← {OP^{(in)Privées}_{Changement}} + {op_i};
                Retourner vrai;
            Fin Si
        Sinon
            Si (P_PCR = {AvecRejet}) Alors
                OP^{in}_{Rejetées} ← {OP^{in}_{Rejetées}} + {op_i};
                Retourner faux;
            Fin Si
        Fin Si
    Sinon
        Pour ∀op_j ∈ OP^{in}_{Différées} faire
            Si (Satisfait(C^{op_j})) Alors
                OP^{in}_{Différées} ← {OP^{in}_{Différées}} - {op_j};
                Notifier(in, in, op_j);
            Fin Si
        Fin Pour
        Si (Satisfait(C^{op_i}) [Condition de report]) Alors
            Si (op_i) Alors
                Si (P^{op_i} = {Globale}) Alors
                    OP^{(in)Privées}_{Cycledevie} ← {OP^{(in)Privées}_{Cycledevie}} + {lop};
                    OP^{(in)Globales}_{Changement} ← {OP^{(in)Globales}_{Changement}} + {op_i};
                    Retourner vrai;
                Sinon
                    OP^{(in)Privées}_{Cycledevie} ← {OP^{(in)Privées}_{Cycledevie}} + {lop};
                    OP^{(in)Privées}_{Changement} ← {OP^{(in)Privées}_{Changement}} + {op_i};
                    Retourner vrai;
                Fin Si
            Sinon
                Si (P_PCR = {AvecRejet}) Alors
                    OP^{in}_{Rejetées} ← {OP^{in}_{Rejetées}} + {op_i};
                    Retourner faux;
                Fin Si
            Fin Si
        Sinon
            OP^{in}_{Différées} ← {OP^{(in}_{Différées}} + {op_i};
            Retourner faux;
        Fin Si
    Fin Si
Fin
```

TAB. B.9 – Algorithme d'intégration d'une opération de changement dans une Incidence.

Données : *Un n-uplet OP d'étiquettes d'opérations* $[Op_{l=1,\ldots,n/(n>2)}]$
Résultat : *Une structure linéaire du graphe des arrangements centré sur le noyau de OP*

Global *résultat* : ***structure linéaire d'opérations*** ;

Procédure Linéariser($[Op_i]$,$[Op_j]$)
  [Ajouter le n-uplet à la structure linéaire résultat]
  Intégrer($[Op_j]$,*résultat*) ;
  Si (Card($[Op_i]$) = 1) **Alors**
    Fin ;
  Sinon
    Pour $\forall\ op^k \in [Op_j]$ **faire**
      Linéariser($[Op_i]$ + op^k, $[Op_j]$ - op^k) ;
      [+ ajoute une opération à un n-uplet selon un ordre lexicographique]
      [- supprime une opération du n-uplet]
    Fin Pour
  Fin Si
Fin
Programme pricipale
  Linéariser([], OP) ;
  Résultat $\leftarrow$ *résultat* ;
Fin

TAB. B.10 – Algorithme récursif de linéarisation du graphe des arrangements centré sur le noyau.

Annexe C

Preuves, définitions et exemples

C.0.1 Définitions et propriété

Définition ***(Estampille logique)***. Une estampille logique $HL(op)$ d'une opération op dans un système d'opérations SO^i est un couple (HL_{SO^i}, SO^i), au sens de *L. Lamport* [53]. On a $(HL_{SO^i}, SO^i) < (HL_{SO^j}, SO^j)$ si et seulement si
- ou bien $HL_{SO^i} < HL_{SO^j}$;
- ou bien $HL_{SO^i} = HL_{SO^j}$ et $SO^i < SO^j$.

HL_{SO^i} désigne l'horloge logique du système d'opération SO^i et EL_{op} désigne l'estampille logique attribuée à l'opération op lors de sa projection :
- si une opération purement privée se produit sur le système d'opération SO^i, HL_{SO^i} est incrémentée ;
- si une opération de projection correspondant à l'envoi d'un message, contenant une ou plusieurs opérations devant être projetées, se produit sur un autre système d'opération SO^i, HL_{SO^i} est incrémentée et le message m est envoyé avec la nouvelle valeur de HL_{SO^i} comme estampille ($EL_m = HL_{SO^i}$) ;
- si une opération correspondant à la réception d'un message m avec une estampille EL_m se produit sur un système d'opérations SO^i, $HL_{SO^i} = \max(HL_{SO^i}, EL_m) + 1$.

Définition ***(Stabilité d'un système d'opérations du flot de contrôle)***. Un système d'opération SO est dit stable si $(OP^{SO}_{Rejetées} = \emptyset \wedge OP^{SO}_{Différées} = \emptyset)$. Il est noté SOS.

Définition ***(Stabilité d'un système d'opérations du flot de contrôle à n-opérations près)***. Deux systèmes d'opérations SO^1 et SO^2 sont stables à n-opérations près si $[((OP^{SO^1}_{Rejetées} \subseteq OP^{SO^2}_{Rejetées}) \vee (OP^{SO^2}_{Rejetées} \subseteq (OP^{SO^1}_{Rejetées})) \wedge (n = card(OP^{SO^1}_{Rejetées} \cap OP^{SO^2}_{Rejetées}))]$. Cela permet de définir des relations entre un noyau et des dérives sans condition de stabilité.

Définition ***(Relation de proportionnalité)***. Soient deux ensembles d'opérations ordonnées $[op^1_i]$ et $[op^2_i]$. On dit que $[op^1_i]$ est proportionnel à $[op^2_i]$, et on note $[op^1_i] \propto [op^2_i]$, si et seulement si $[op^2_i] \subseteq [op^1_i]$.

Définition ***(Proportionnalité des systèmes d'opérations purgés)***. Soient deux systèmes d'opérations purgés SOP^1 et SOP^2. On dit que SOP^1 est proportionnel à SOP^2, et on note $SOP^1 \propto SOP^2$, si et seulement si $(\forall op^{Changement}_i \in SOP^2 \Rightarrow op^{Changement}_i \in SOP^1)$. On ne considère ici que les opérations

de changement, car il doit être possible d'évaluer la proportionnalité entre une instance et une stratégie, puisque, cette dernière ne possède pas d'opérations de cycle de vie sur les activités.

Définition ***(Proportionnalité des systèmes d'opérations du flot de contrôle)***. Soient deux systèmes d'opérations SO^1 et SO^2. Ils sont proportionnels, et on note $SO^1 \propto SO^2$ si
- SO^1 dérive de SO^2 ;
- et $SOP^{SO^1} \propto SOP^{SO^2}$;
- et SO^1 et SO^2 sont stables à n-opérations près.

Définition ***(Analogie des systèmes d'opérations du flot de contrôle)***. Deux systèmes d'opérations SO^1 et SO^2 sont dits analogues (proportionnels par le noyau), et on note $SO^1 \propto^* SO^2$, si et seulement si, $\exists\ SO^3$ tel que :
- $SO^1 \propto SO^3$;
- et $SO^2 \propto SO^3$.

Définition ***(Écart entre systèmes d'opérations)***. L'écart entre deux systèmes d'opérations SO^1 et SO^2, noté $\bowtie_{SO^2}^{SO^1}$, est défini par $\{\forall op_i/(op_i \in \{SOP_{Changement}^{SO^1}\} - \{SOP_{Changement}^{SO^2}\}) \vee (op_i \in \{SOP_{Changement}^{SO^2}\} - \{SOP_{Changement}^{SO^1}\})\}$. On a aussi $\bowtie_{SO^2}^{SO^1} = \bowtie_{SO^1}^{SO^2}$.

Propriété ***(Δ-rayon)***. Un $\mathbb{K}^\Delta$ avec $\Delta = [op_{i=1...n}]$ possède un delta-rayon, noté R_Δ, qui est définit par $R_\Delta = card(\Delta) = n$ (se reporter à la figure C.4).

Les figures A.6, A.7, A.9 et A.11 illustrent la construction d'un $\mathbb{K}^\Delta$ avec un $\Delta - rayon$ allant de 1 à 4.

Propriété ***(Cardinalité d'un nœud)***. Un nœud $N_i = [op_l, ..., op_k]$ d'un $\mathbb{K}^\Delta$ à une caridinalité, noté r_Δ, représentant sa taille en nombre d'opérations, et est définie par $r_\Delta = card([op_l, ..., op_k]) = (k - l + 1)$ tel que $(r_\Delta \leq R_\Delta etl \geq 1)$ (se reporter à la figure C.4).

Propriété ***(Excentricité d'un nœud)***. Un nœud $N_i = [op_l, ..., op_k]$ d'un $\mathbb{K}^\Delta$ à une excentricité, noté d_{op}, représentant sa distance par rapport au nœud-noyau, et est définie par $d_{op} = R_\Delta - r_\Delta$ (se reporter à la figure C.4).

Propriété ***(L'ordre d'un $\mathbb{K}^\Delta$)***. Le nombre de sommets dans un $\mathbb{K}^\Delta$ est égale à

$$[R_\Delta \times \sum_{i=0}^{R_\Delta - 2} A_{R_{\Delta-1}}^i] + 1$$

Propriété ***(Représentation d'un $\mathbb{K}^\Delta$)***. Un $\mathbb{K}^\Delta$ est un graphe planaire, c'est-à-dire, qu'il peut être dessiné sans que ses arcs ne se croissent (voir les figures A.6, A.7, A.9 et A.11).

Définition ***(Degré d'un nœud de $\mathbb{K}^\Delta$)***. Le degré d'un nœud $N_{r_\Delta} = [op_1, ..., op_{r_\Delta}]$ d'un $\mathbb{K}^\Delta$, à (pour $R_\Delta \geq 2$) :

- un degré extérieur égale à $(R_\Delta - r_\Delta)$, tel que $(1 \leq r_\Delta \leq R_\Delta)$;
- un degré extérieur égale à

  1 si $(1 < r_\Delta < R_\Delta)$

  0 si $(r_\Delta = 1)$

$(R_\Delta!)$ si $(r_\Delta = R_\Delta \wedge R_\Delta > 1)$
0 sinon.

Définition ***(Degré d'un nœud de*** $\mathbb{N}^\Delta$***)***. Le degré d'un nœud $N_{r_\Delta} = [op_1, ..., op_{r_\Delta}]$ d'un $\mathbb{N}^\Delta$ possède :
- un degré extérieur égale à
  $(R_\Delta - r_\Delta)$ si $(R_\Delta > 1 \wedge r_\Delta = 1 \wedge N_{r_\Delta} = [op_1] = \Delta_1)$
  $(R_\Delta!)$ si $(R_\Delta > 1 \wedge r_\Delta = R_\Delta)$
  $(R_\Delta - r_\Delta) + 1$ si $((R_\Delta > 1 \wedge r_\Delta = 1) \vee 1 \leq r_\Delta \leq (R_\Delta - 1))$
  1 si $(R_\Delta > 1 \wedge r_\Delta = R_\Delta - 1)$
  0 sinon.
- un degré extérieur égale à
  $(R_\Delta - r_\Delta)$ si $((r_\Delta = 1 \wedge N_{r_\Delta} = [op_1] = \Delta_{R_\Delta}) \vee (r_\Delta = R_\Delta))$
  $(R_\Delta - r_\Delta) + 1$ sinon

C.0.2 Preuve de T_1

Il s'agit ici de prouver que l'opération $[op :: InsérerSérie(I, a_{ins}, a_{pré}, a_{post})]$ est correcte $\Leftrightarrow [(\{\text{Créer}(a_{pré})\}, \{\text{Créer}(a_{post})\} \in SOP^I) \wedge (\{\text{Créer}(a_{ins})\}, \{\text{Activer}(a_{post})\}, \{\text{Abandonner}(a_{pré})\}, \{\text{Terminer}(a_{pré})\} \notin SOP^I)]$

Soient les propositions P_1, P_2, P_3, P_4, P_5, P_6 et ISC tels que

- $ISC \equiv \{InsérerSérie(I, a_{ins}, a_{pré}, a_{post})]$ est correcte$\}$;
- $P_1 \equiv \{Créer(a_{pré}) \in SOP^I\}$;
- $P_2 \equiv \{Créer(a_{post}) \in SOP^I\}$;
- $P_3 \equiv \{Créer(a_{ins}) \notin SOP^I\}$;
- $P_4 \equiv \{Activer(a_{post}) \notin SOP^I\}$;
- $P_5 \equiv \{Abandonner(a_{pré}) \notin SOP^I\}$;
- $P_6 \equiv \{Terminer(a_{pré}) \notin SOP^I\}$.

Prouver l'équivalence revient à prouver que

$ISC \Leftrightarrow P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6$

Tout d'abord, montrons « $\Rightarrow$» et que :

$ISC \Rightarrow P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6 \wedge P_7$

Prouvons que $\neg\ (P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6) \wedge ISC \equiv Faux$

On a $\neg\ (P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6) \wedge ISC \equiv$

$$(\neg P_1 \wedge ISC) \vee (\neg P_2 \wedge ISC) \vee (\neg P_3 \wedge ISC) \vee (\neg P_4 \wedge ISC) \vee (\neg P_5 \wedge ISC) \vee (\neg P_6 \wedge ISC)$$

Or

- $(\neg P_1 \wedge ISC) \equiv Faux$, car l'insertion ne peut avoir lieu sans l'existante de l'activité $a_{pré}$;
- $(\neg P_2 \wedge ISC) \equiv Faux$, car l'insertion ne peut avoir lieu sans l'existante de l'activité a_{post} ;
- $(\neg P_3 \wedge ISC) \equiv Faux$, car l'insertion ne peut avoir lieu si l'activité a_{ins} est déjà présente ;
- $(\neg P_4 \wedge ISC) \equiv Faux$, car l'insertion ne peut avoir lieu si l'activité a_{post} est déjà activée ;
- $(\neg P_5 \wedge ISC) \equiv Faux$, car l'insertion ne peut avoir lieu dans une branche de flot abandonnée ;
- $(\neg P_6 \wedge ISC) \equiv Faux$, car l'insertion ne peut avoir lieu dans le cas où $a_{pré}$, c-à-d, que l'insertion est concurrente à l'opération d'évaluation de la transition entre les activités $a_{pré}$ et a_{post}.

Par conséquent $\neg\ (P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6) \wedge ISC \equiv Faux$

C.Q.F.D pour « $\Rightarrow$»

Maintenant, montrons « $\Leftarrow$» :

Soient les propositions ISC, P_7, P_8, P_9 tels que

- $ISC \equiv \{InsérerSérie(I, a_{ins}, a_{pré}, a_{post})]$ est correcte$\} \equiv \{Créer(a_{ins}) \in SOP^I\}$;
- $P_7 \equiv \{\forall a \in ASucc^{a_{ins}}/Activer(a) \notin SOP^I\}$;
- $P_8 \equiv \{\forall a \in APred^{a_{ins}}/Terminer(a) \notin SOP^I\}$;
- $P_9 \equiv \{\forall a \in APred^{a_{ins}}/Abandonner(a) \notin SOP^I\}$.

Montrons que $P_7 \wedge P_8 \wedge P_9 \Rightarrow ISC$

Pour cela prouvons que

$\neg(P_7 \wedge P_8 \wedge P_9) \wedge ISC \equiv Faux$

Comme $\neg(P_7 \wedge P_8 \wedge P_9) \wedge ISC \equiv$

$$(\neg P_7 \wedge ISC) \vee (\neg P_8 \wedge ISC) \vee (\neg P_9 \wedge ISC)$$

Or

- $(\neg P_7 \wedge ISC) \equiv \{\exists a \in ASucc^{a_{ins}}/\ [Activer(a), Créer(a_{ins})] \in SOP^I\} \equiv Faux$, correspondant à une insertion dans une branche activée du flot ;
- $(\neg P_8 \wedge ISC) \equiv \{\exists a \in APred^{a_{ins}}/\ [Terminer(a), Créer(a_{ins})] \in SOP^I\} \equiv Faux$, correspondant à une insertion dans une branche terminée du flot ;
- $(\neg P_9) \wedge ISC) \equiv \{\exists a \in APred^{a_{ins}}/\ [Abandonner(a), Créer(a_{ins})] \in SOP^I\} \equiv Faux$, correspondant à une insertion dans une branche abandonnée du flot.

$\neg(P_7 \wedge P_8 \wedge P_9) \wedge ISC \equiv Faux$

C.Q.F.D pour « $\Leftarrow$»

Et finalement nous avons bien T_1.

Cette preuve du théorème reste, à notre avis, peu satisfaisante car il faudrait utiliser les techniques avancées de preuves de théorèmes pour enfin pouvoir l'admettre. La démonstration des autres théorèmes T_2 et T_3 est identique, sauf que pour le cas de la suppression, il faut considérer que l'histoire ne peut être remise en cause et donc on ne peut supprimer une activité appartenant à une branche activée, terminée

ou abandonnée. La preuve de T_4, T_5 T_6 est la même car il ne s'agit que d'une restriction sur certaine opérations.

C.0.3 Exemples

Le tableau suivant est un exemple du passage d'une incidence I réduite du tableau C.2 (voir plus loin) à une incidence purgé.

SOR^I	SOP^I
Créer(I,$a_{initiale}$) ;	Créer(I,$a_{initiale}$) ;
Créer(I,a_{finale}) ;	Créer(I,a_{finale}) ;
InsérerSérie(I,X,$a_{initiale}$,a_{finale}) ;	InsérerSérie(I,X,$a_{initiale}$,a_{finale}) ;
Créer(I,X) ;	Créer(I,X) ;
Activer(I,$a_{initiale}$) ;	Activer(I,$a_{initiale}$) ;
Activer(I,X) ;	Activer(I,X) ;
Exécuter(I,X) ;	Exécuter(I,X) ;
InsérerSérie(I,Y,X,a_{finale}) ;	-
Créer(I,Y) ;	-
Supprimer(I,Y) ;	-
InsérerSérie(I,Y,X,a_{finale}) ;	InsérerSérie(I,Y,X,a_{finale}) ;
Créer(I,Y) ;	Créer(I,Y) ;
Terminer(I,X) ;	Terminer(I,X) ;
Activer(I,Y) ;	Activer(I,Y) ;
Exécuter(I,Y) ;	Exécuter(I,Y) ;
Terminer(I,Y) ;	Terminer(I,Y) ;
Terminer(I,a_{finale})	Terminer(I,a_{finale}) ;

TAB. C.1 – Exemple explicitant les relations entre SOR^I et SOP^I d'une incidence I.

Ce tableau montre la réduction d'un système d'opérations avec un exemple d'exécution d'une incidence I. Se reporter à la section 3.2.

OP^I	SOR^I
Créer(I) ;	-
Créer(I,S,$a_{initiale}$) ;	Créer(I,$a_{initiale}$) ;
Créer(I,a_{finale}) ;	Créer(I,a_{finale}) ;
Créer(I,$t_{initiale \to finale}$,$a_{initiale}$,a_{finale}) ;	-
Créer(I,[InsérerSérie(X,$a_{initiale}$,a_{finale})]) ;	-
Intégrer(I,[InsérerSérie(X,$a_{initiale}$,a_{finale})]) ;	-
InsérerSérie(I,X,$a_{initiale}$,a_{finale}) ;	InsérerSérie(I,X,$a_{initiale}$,a_{finale}) ;
Notifier([InsérerSérie(I,X,$a_{initiale}$,a_{finale})]) ;	-
Créer(I,X) ;	Créer(I,X) ;
Supprimer(I,$t_{initiale \to finale}$) ;	-
Créer(I,$t_{initiale \to X}$,$a_{initiale}$,X) ;	-
Créer(I,$t_{X \to finale}$,X,a_{finale}) ;	-
Activer(I) ;	-
Activer(I,$a_{initiale}$) ;	Activer(I,$a_{initiale}$) ;
ActiveràVrai(I,$t_{initiale \to X}$) ;	-
Activer(I,X) ;	Activer(I,X) ;
Exécuter(I,X) ;	Exécuter(I,X) ;
Intégrer(I,[InsérerSérie(Y,X,a_{finale})]) ;	-
InsérerSérie(I,Y,X,a_{finale}) ;	InsérerSérie(I,Y,X,a_{finale}) ;
Notifier([InsérerSérie(Y,X,a_{finale})]) ;	-
Créer(I,Y) ;	Créer(I,Y) ;
Créer(I,$t_{X \to Y}$,X,Y) ;	-
Créer(I,$t_{Y \to finale}$, Y, a_{finale}) ;	-
Intégrer(I,[Supprimer(I,Y)]) ;	-
Supprimer(I,Y) ;	Supprimer(I,Y) ;
Notifier([Supprimer(Y)]) ;	-
Supprimer(I,$t_{X \to Y}$) ;	-
Supprimer(I,$t_{Y \to finale}$) ;	-
Intégrer(I,[InsérerSérie(Y,X,a_{finale})]) ;	-
InsérerSérie(I,Y,X,a_{finale}) ;	InsérerSérie(I,Y,X,a_{finale}) ;
Notifier ([InsérerSérie(Y,X,a_{finale})]) ;	-
Créer(I,Y) ;	Créer(I,Y) ;
Créer(I,$t_{X \to Y}$, X, Y) ;	-
Créer(I,$t_{Y \to finale}$, Y, a_{finale}) ;	-
Terminer(I,X) ;	Terminer(I,X) ;
ActiveràVrai(I,$t_{X \to Y}$) ;	-
Activer(I,Y) ;	Activer(I,Y) ;
Exécuter(I,Y) ;	Exécuter(I,Y) ;
Terminer(I,Y) ;	Terminer(I,Y) ;
ActiveràVrai(I,$t_{Y \to finale}$) ;	-
Terminer(I,a_{finale})	Terminer(I,a_{finale}) ;

TAB. C.2 – Exemple explicitant les relations entre SO et SOR^I d'une incidence I.

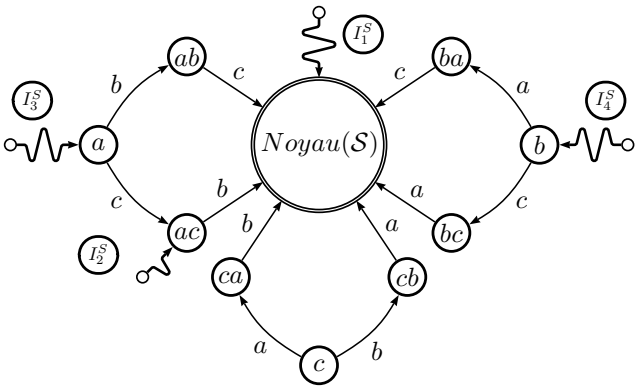


FIG. C.1 – Illustration d'un $\circledcirc^{[a,b,c]}$.

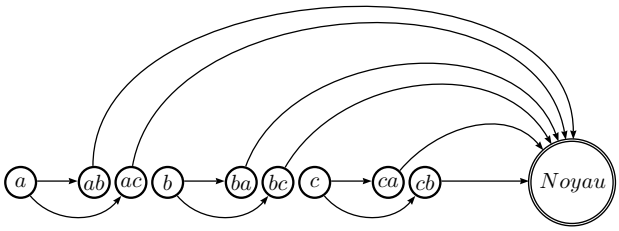


FIG. C.2 – Illustration simplifiée d'un $\circledcirc^{[a,b,c]}$.

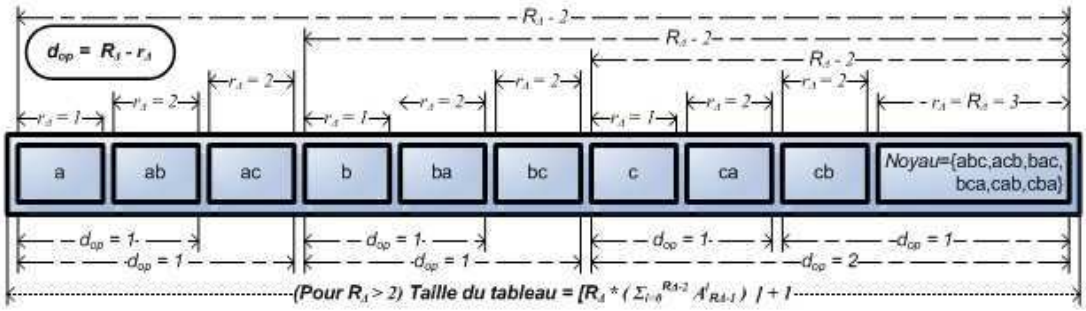


FIG. C.3 – Structure linéaire représentant un $\mathbb{K}^{[a,b,c]}$.

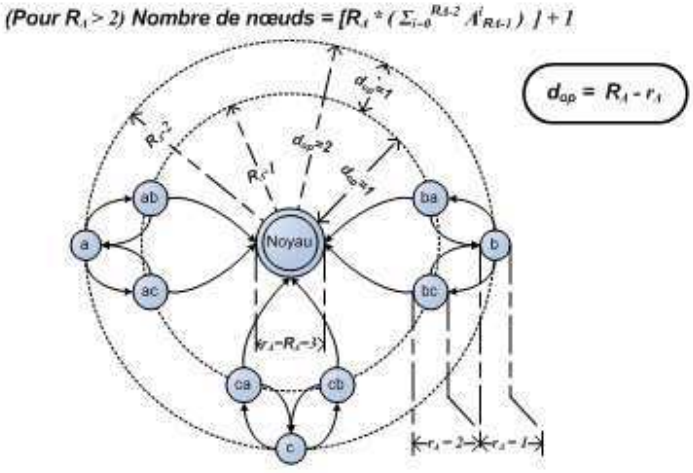


FIG. C.4 – Illustration des propriétés d'un $\mathbb{K}^{[a,b,c]}$.

Annexe D

L'origine du changement : les bases de données orientées objets

La problématique de l'**évolution du schéma** (*Schema Evolution*) a été longtemps une discipline du domaine des systèmes de gestion des bases de données orientées objets [54, 55, 56]. La première tentative de résolution de cette problématique se trouve dans [54]. Une autre tentative dans [55], plus étoffée, présente une taxonomie complète des opérations de modification du schéma. Les règles proposées posent des contraintes sur l'évolution du schéma. Ces règles préservent ainsi **les invariants** du schéma, c'est-à-dire, des conditions sur le schéma qui doivent être satisfaites avant et après la modification [10]. La mise en œuvre emploie des techniques de programmation orientée aspects pour fournir un moyen flexible pour transformer des objets vis-à-vis de leur évolution en appliquant des primitives de transformation. Ceci permet d'éviter **l'érosion de la structure** de la base de données par le maintien d'un historique cohérent et compréhensible des changements survenus.

Historiquement, la communauté des bases de données a employé fondamentalement les trois techniques suivantes de modification de la structure conceptuelle d'une base de données orienté objets [54] :

- **Gestion de l'évolution du schéma** (*Schema Evolution*), par exemple [57, 58] : selon laquelle une base de données est représentée par un schéma logique sur lequel sont appliquées les modifications des définitions des classes et de leur hiérarchie ;
- **Gestion des versions de classes** (*Class Versioning*), par exemple [59, 60, 61] : laquelle maintient les différentes versions de chaque type de données, et renvoie ainsi chaque instance d'objet à une version spécifique du type ;
- **Gestion des versions du schéma** (*Schema Versioning*), par exemple [62, 63] : permettant de créer différentes versions d'un même schéma logique, et acceptant d'effectuer des modifications indépendantes sur ces versions.

Outre la nécessité de gérer l'évolution de leurs définitions (c'est-à-dire, les versions de classes), d'autres stratégies, par exemple [64, 65], ont permis la **gestion des versions des objets** résidant dans la base de données.

Ces différentes approches se focalisent sur un seul aspect de l'évolution au lieu de considérer l'évolution comme un processus qui affecte l'ensemble de la base. Explicitement, la catégorie des approches pour gérer l'évolution du schéma essaye de fournir à chaque instant une vue cohérente et compréhensible du système. Pour cela, elles emploient des fonctions de transformations par défaut ou définies par l'utilisateur, appliquées instantanément ou en différé, pour maintenir les instances d'objets en conformité avec le schéma modifié. Toutefois, les informations essentielles sur l'historique de l'évolution sont perdues, ce qui

est pénalisant si le changement doit être annulé. D'autre part, la gestion de versions de classes garde l'historique des transformations de la définition des classes. Ce qui ne nécessite point de changer les objets existants pour refléter les changements survenus au niveau des classes qu'ils instancient. Malgré cela, comme le nombre de versions pour chaque classe s'accroît, le schéma tend à devenir plus complexe, rendant la maintenance assez difficile. Dans ce cas de figure, il est également laborieux d'obtenir une vision cohérente du système avec la présence d'un grand nombre de versions de classe(s).

La gestion des versions du schéma essaye de pallier aux lacunes des deux approches précédentes en offrant une vue cohérente de la structure conceptuelle de la base de données, tout en maintenant des informations sur l'évolution de ce schéma. L'historique est, cependant, à une granularité moins fine que dans les versions de classes. Bien qu'elle soit particulièrement utile pour le maintien de la compatibilité d'avant et d'après le changement avec les applications existantes, cette technique est onéreuse en terme d'utilisation de l'espace, en particulier en présence d'un grand nombre de versions de schéma lorsque des modifications mineures d'une classe conduisent à la création d'une nouvelle version du schéma.

Dans [54], les auteurs présentent une approche composite permettant de résoudre le problème de l'évolution des différents types d'objets contenus dans le dictionnaire, ainsi que le schéma de la base de données elle-même. Cette approche considère l'évolution comme un processus affectant la totalité de la base de données comme « l'évolution de la base de données» (*DataBase Evolution*). Elle allie les deux techniques de gestion de l'évolution du schéma, ainsi que celle de la gestion des versions de classes pour donner une approche composite.

Annexe E

La théorie du changement dans les Workflows

Au niveau du modèle du Workflow, les opérations de changements modifient un schéma de procédé en altérant un ensemble d'activité ou en changeant leur ordre. Ainsi, la modification d'un schéma produit un schéma différent. Selon le métamodèle que nous utilisons (se reporter à la section 1.1.1), un schéma de procédé peut être formellement décrit sans nécessiter une notation graphique particulière. Pour cela, nous faisons abstraction des opérateurs d'un langage de modélisation concret de procédés et nous décrivons seulement l'ensemble des activités et leur comportement possible.

Schéma de procédé : Un schéma de procédé est un couple $SP = (A, TS)$ [26] où,

- A est un ensemble d'activités ;
- $TS = (S, T, S_{début}, S_{fin})$ est un réseau de transitions étiquetées, où S est l'ensemble des états atteignables, $T \subseteq S \times (A \cup \tau)$ est une relation de transition, $S_{début} \in S$ est l'état initial, et $S_{fin} \in S$ est l'état final.

P étant l'ensemble de tous les schémas de procédé.

Il est à noter que tout langage de modélisation de processus métier peut être transcrit [66] en un réseau de transitions étiquetées TS. Ce système de transitions ne définit pas seulement l'ensemble des traces possibles (c'est-à-dire, l'ordre d'exécution), mais exprime aussi les moments ou une sélection se produit dans le flot de contrôle. Toutefois, il autorise des **transitions muettes**. Une transition muette, notée « τ» , est une activité qui change l'état du réseau (le modèle ou l'instance) du procédé, mais qui n'est cependant pas observable dans l'historique d'exécution. De ce fait, on peut distinguer entre différents types de choix (interne/externe ou contrôlable/incontrôlable) [66]. Si toutes les opérations de changement modifient l'ensemble des états S et la relation de transition T, l'opération de déplacement est la seule qui n'a d'effets que sur l'ensemble A des activités.

Pour pouvoir comparer des séquences d'opérations de changement, et dériver une relation d'ordre entre ces changements, il est nécessaire de définir une relation d'équivalence entre les schémas de procédés. Il existe de nombreux concepts d'équivalence entre procédés. La notion la plus élémentaire est celle de l'**équivalence de traces d'exécution** (voir section 2.3) caractérisant deux schémas de procédés équivalents par une évaluation de la similarité entre des observations de leurs historiques d'exécution. Puisque le nombre de traces générées par un modèle de Workflow peut être infini (car il n'y a pas de limite sur le nombre d'instanciations), une telle comparaison s'avère compliquée à réaliser. De plus, se limiter à comparer l'équivalence des traces d'exécution peut faillir à saisir le moment où des sélections se produisent dans le procédé [67]. Par exemple, deux schémas de procédés (au travers de leurs instances)

peuvent produire le même ensemble de traces $\{ABC, ABD\}$. Quoique, le procédé pourrait exprimer des différences lors des choix sur les conditions, c'est-à-dire, que le premier procédé pourrait avoir le choix d'exécuter soit BC ou BD après l'activité A, alors que le deuxième aurait le choix d'exécuter soit C ou D juste après l'activité B. La **bissimilarité des ramifications** est un autre exemple d'équivalence qui peut correctement capturer les instants des sélections. Dans ce mémoire, nous faisons abstraction de toute notion concrète d'équivalence pour ne pas se limiter à une notation particulière.

Comme, il est évoqué plus haut, toute application d'une opération de changement transforme un schéma de procédé en un schéma différent. Ceci peut être formalisé comme suit :

Le changement dans un schéma de procédé : Soient SP_1, $SP_2 \in P$ deux schémas de procédés et soit Δ un changement du schéma,

- $SP_1 \rightsquigarrow_\Delta$ existe, si et seulement si Δ est applicable à SP_1, c'est-à-dire que Δ est réalisable dans SP_1 ;
- $SP_1 \rightsquigarrow_\Delta SP_2$ exite, si et seulement si Δ est applicable à SP_1 (c'est-à-dire, $SP_1 \rightsquigarrow_\Delta$) et SP_2 est le schéma de procédé résultant de l'application de Δ à SP_1.

La réalisation du changement à un schéma de procédé est définie dans [26]. Par exemple dans le cas de la figure E.3 de l'annexe E, une activité E ne peut être insérée dans un schéma S, entre deux ensembles d'activités $\{1\}$ et $\{2\}$, que si ces ensembles sont contenus dans S et l'activité E n'est pas dans S. Il est à préciser, que la duplication des activités n'est pas admise, c'est-à-dire, que l'occurrence d'une activité dans un schéma ne se produit qu'une seule fois.

Sur la base de l'équivalence entre schémas, il est possible de définir la notion de commutativité entre les opérations de changement.

Commutativité du changement : Soit $SP \in P$ un schéma de procédé, et soient Δ_1 et Δ_2 deux changements de procédé. Δ_1 et Δ_2 commutent (voir figure E.1) dans SP si et seulement si :

- Il existe SP_1, $SP_2 \in P$, tels que $SP \rightsquigarrow_{\Delta_1} SP_1$ et $SP_1 \rightsquigarrow_{\Delta_2} SP_2$;
- Il existe SP_3, $SP_4 \in P$, tels que $SP \rightsquigarrow_{\Delta_2} SP_3$ et $SP_3 \rightsquigarrow_{\Delta_1} SP_4$;
- $SP_2 \equiv SP_4$.

Deux opérations de changement sont commutatives, si elles ont le même effet sur le schéma, indépendamment de l'ordre selon lequel elles sont appliquées (la figure E.1 schématise la relation de commutativité). Par contre, si deux opérations de changement ne sont pas commutatives, elles sont considérées comment dépendantes, c'est-à-dire, que l'effet de l'une dépendant de l'effet de l'autre. Et elles doivent être appliquées selon leur ordre d'occurrence. Selon la nature du patron du changement, il parfois nécessaire d'appliquer la série des opérations de changement atomique $op_{i=1,...,n}$ comme une transaction $\Delta = (op_1, ..., op_n)$ pour garantir la correction de cette modification.

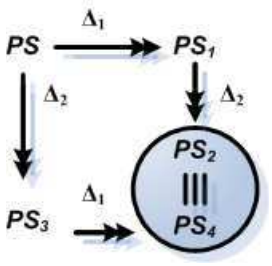


FIG. E.1 – Relation de commutativité du changement.

Pour illustrer la propagation des changements du schéma (c'est-à-dire, du modèle de procédé) au niveau opérationnel (autrement dit, aux instances de ce procédé) aussi bien à des **instances dites « biaisées »** (ayant dérivée du schéma initial par des changements individuels d'une manière *ad-hoc*) qu'à des **instances « non biaisées »** (conforment au schéma initial) nous utilisons le premier axiome de [23].

Propagation des changements du schéma aux instances biaisées : Soit S une version du schéma selon un modèle de procédé T. Supposons qu'une nouvelle version (correcte) S' du schéma soit dérivée de S par application d'un changement du schéma Δ_T. C'est alors que Δ_T pourrait être propagé à une instance I (conforme au modèle T, qui s'exécute selon un schéma $S_I := S + \delta_I$) : $\Leftrightarrow$

- ***Correction structurelle*** : $S_I^* := (S + \delta_I) + \Delta_T$ est un schéma correct selon le critère de correction structurelle défini sur le métamodèle du Workflow utilisé (voir section 1.1.5). Explicitement, Δ_T peut être correctement appliqué à $S_I = (S + \delta_I)$;
- ***Correction relative à l'état*** : I est conforme avec S_I^*, c'est-à-dire, selon le critère de conformité utilisé ;
- ***Correction sémantique*** : Δ_T et δ_I ne sont pas en conflit sémantique.

La figure 1.3 illustre le fait qu'il n'est pas toujours évident d'appliquer un changement du schéma à une instance biaisée du même procédé.

En contraste avec l'expressivité des langages de Workflow qui détermine leur potentiel à modéliser des changements complexes [6]. Les auteurs de [68] affirment : « *qu'en générale, plus le langage de modélisation des procédés est expressif (c'est-à-dire, que le support à mieux exprimer le flot de contrôle et le flot de données est plus important), plus le changement devient complexe et difficile à réaliser* ». L'existence de plusieurs paradigmes de modélisation et le manque de méthodes de comparaison des différentes approches pour réaliser les changements, font qu'il est difficile pour les ingénieurs de PMS de choisir une technologie adéquate.

À la différence des **patrons de gestion des exceptions** (*Exception Handling Patterns*) [69] qui ne permettent que des changements comportementaux (par exemple, le patron *Rollback* ne change que l'état d'une instance du procédé) mais non du schéma, [68] proposent 17 **patrons du changement** (*Change Patterns*) et 6 **techniques de support au changement** (*Change Support Features*), qui en combinaison permettent d'évaluer les technologies de gestion des procédés existantes vis-à-vis de leur respect du support au changement. L'introduction de la notion de patron du changement n'a pour but que la comparaison. Les patrons ont été, pour la première fois, introduits par *Ch. Alexander* comme une description de solutions pour des problèmes récurrents [70], et leur grand succès, initier par le *GOF* [52] et D. *Schmidt* [71], dans le génie logiciel qu'on leur connaît. Dans le domaine des WfMSs, les patrons [15] ont été proposés pour l'analyse de l'expressivité du BPM (*Control-Flow Patterns*) [19] et la description des différentes manières de modéliser les aspects liés aux données (*Workflow Data Patterns*) [18] et aux ressources (*Workflow Ressource Patterns*) [17] dans les BPML. Cependant, pour une évaluation du potentiel d'un PMS selon sa flexibilité et son adaptation aux changements, ces patrons sont importants mais insuffisants. D'autant plus est, qu'un ensemble de patrons du changement offre un indicateur intéressent pour savoir si un *framework* (du changement) permet d'exprimer des modifications plus ou moins complexes dans le flot de contrôle des procédés. Les patrons d'adaptation [68] réduisent la complexité de l'évolution des procédés (explicitement, ce que sont les patrons de conception- *Design Patterns*- au génie logiciel) et élèvent le niveau d'expression des changements en faisant abstraction des opérations s'appliquant aux nœuds et aux transitions du modèle de Workflow. En conséquence, les primitives d'évolution du Workflow ne sont plus considérées que comme des opérations de bas niveau. Indépendamment qu'ils soient réalisés sur le schéma ou les instances du procédé, c'est à ce niveau haut, que nous illustrons les principaux changements que doit pouvoir supporter et offrir un PMS.

- L'insertion d'un fragment de procédé (voir les figures E.3, E.4, et E.5) ;
- La suppression d'un fragment de procédé (voir figure E.6) ;
- Le déplacement peut être réalisé sous forme de suppression et d'insertion de fragment de procédé

(voir figure E.7) ;
– Le remplacement d'un fragment de procédé (voir figure E.8).

D'autres patrons peuvent être employés, par exemple : échange d'un ensemble d'activités, extraction ou inclusion d'un sous-procédé à partir d'un schéma, encastrement d'un fragment de procédé dans une boucle, paralléliser des fragments de procédé, ajout ou suppression d'une dépendance de contrôle, modification d'une transition conditionnelle [72].

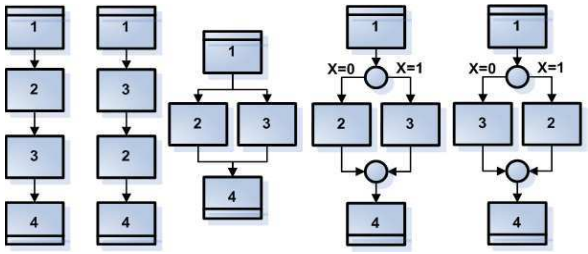


FIG. E.2 – Exemples de changements : modification de l'ordre d'exécution des activités.

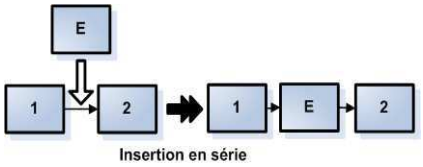


FIG. E.3 – Insertion en série du fragment E entre deux ensembles d'activités consécutifs.

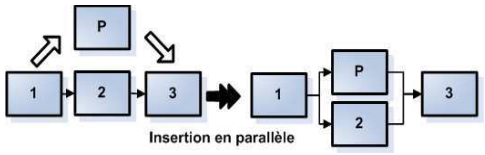


FIG. E.4 – Insertion en parallèle du fragment P entre deux ensembles d'activités sans condition.

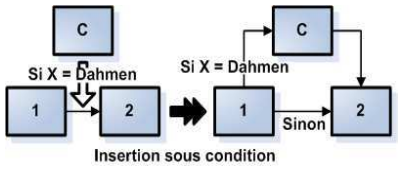


FIG. E.5 – Insertion du fragment C entre deux ensemble d'activités avec une condition.

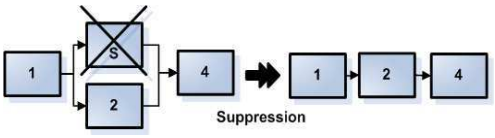


FIG. E.6 – Suppression d'un fragment de procédé S.

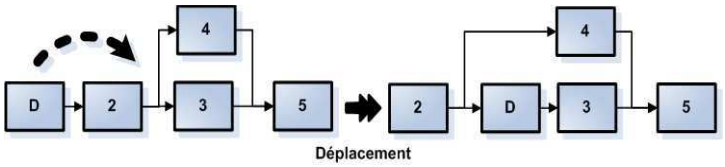


FIG. E.7 – Déplacement d'un fragment de procédé D.

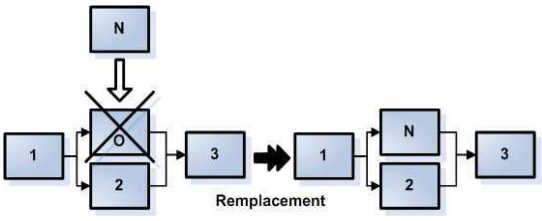


FIG. E.8 – Remplacement d'un fragment O par un fragment de procédé N.

Bibliographie

[1] Bull Corporation. Flowpath functional specification. Bull S. A., September 1992.

[2] M. Dumas *et al.*. *Process Aware Information Systems : Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005.

[3] M. Reichert *et al.*. Adept flex - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2) :93–129, 1998.

[4] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *HICSS '01 : Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 7*, page 7051, 2001.

[5] W. M. P. van der Aalst et *al.*. Case handling : A new paradigm for business process support. *Data and Knowledge Engineering, 53(2) :129–162,*, 2005.

[6] C. A. Ellis *et al.*. Dynamic change within workflow systems. In *COCS '95 : Proceedings of conference on Organizational computing systems*, pages 10–21. ACM Press, 1995.

[7] W. M. P. van der Aalst. Exterminating the dynamic change bug : A concrete approach to support workflow change. *Information Systems Frontiers*, 2001.

[8] A. Agostini *et al.*. Improving flexibility of workflow management systems. *Business Process Management*, pages 289–342, 2000.

[9] F. Casati *et al.*. Workflow evolution. *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 438–455, 1996.

[10] M. Kradolfer *et al.*. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *Conference on Cooperative Information Systems*, pages 104–114, 1999.

[11] S. W. Sadiq *et al.*. Managing change and time in dynamic workflow processes. *International Journal of Cooperative Information Systems*, 9 :93–116, 2000.

[12] S. Rinderle *et al.*. Correctness criteria for dynamic changes in workflow systems–a survey. *Data & Knowledge Engineering*, 50(1) :9–34, July 2004.

[13] W. M. P. van der Aalst. Generic workflow models : How to handle dynamic change and capture management information ? In *Conference on Cooperative Information Systems*, pages 115–126, 1999.

[14] P. Wohed *et al.*. Pattern based analysis of bpel4ws. *QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane*, 2002.

[15] Workflow Patterns Initiative. Documentation, key papers, 2007.

[16] W. M. P. van der Aalst. Patterns and xpdl : A critical evaluation of the xml process definition language. Technical report, Queensland University of Technology, 2003.

[17] N. Russell *et al.*. Workflow resource patterns. *BETA Working Paper Series, WP 127*, 2004.

[18] N. Russell *et al.*. Workflow data patterns. Technical report, Brisbane, 2004.

[19] N. Russell *et al.*. Workflow control-flow patterns : A revised view. *BPM Center Report BPM-06-22*, 2006.

[20] D. Hollingsworth. *The Workflow Reference Model, Document Status, Issue 1.1.* Workflow Management Coalition, 1995.

[21] W. Sadiq *et al.*. On correctness issues in conceptual modeling of workflows. *In Proceedings of the 5 th European Conference on Information Systems (ECIS '97), Cork, Ireland, June 19-21*, 1997.

[22] S. W. Sadiq. Handling dynamic schema change in process models. In *Australasian Database Conference*, pages 120–126, 2000.

[23] S. Rinderle *et al.*. On dealing with structural conflicts between process type and instance changes. *Business Process Management*, pages 274–289, 2004.

[24] S. Rinderle *et al.*. On dealing with semantically conflicting business process changes. *Technical Report UIB-2003-04. University of Ulm, Computer Science Faculty, June 2003.*

[25] L. Ly *et al.*. Semantic correctness in adaptive process management systems. *Business Process Management*, pages 193–208, 2006.

[26] C. Günther *et al.*. Change mining in adaptive process management systems. *On the Move to Meaningful Internet Systems 2006 : CoopIS, DOA, GADA, and ODBASE*, pages 309–326, 2006.

[27] W. M. P. van der Aalst *etal.*. Inheritance of workflows : an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1–2) :125–203, 2002.

[28] D. Georgakopoulos *et al.*. An overview of workflow management : From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2) :119–153, 1995.

[29] C. Mohan *et al.*. State of the art in workflow management research and products. *SIGMOD Rec.*, 25(2), June 1996.

[30] F. Charoy *et al.*. A dynamic workflow management system for coordination of cooperative activities. *Business Process Management Workshops*, pages 205–216, 2006.

[31] M. Pesic *et al.*. A declarative approach for flexible business processes management. *Business Process Management Workshops*, pages 169–180, 2006.

[32] S. Rinderle *et al.*. Flexible support of team processes by adaptive workflow systems. *Distrib. Parallel Databases*, 16(1) :91–116, July 2004.

[33] M. Weske. Object-oriented design of a flexible workflow management system. In *ADBIS '98 : Proceedings of the Second East European Symposium on Advances in Databases and Information Systems*, pages 119–130, London, UK, 1998. Springer-Verlag.

[34] M. Weske. *Business Process Management :Concepts, Languages, Architectures.* 978-3-540-73521-2. 2007.

[35] M. Weske. Flexible modeling and execution of workflow activities. In *HICSS '98 : Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 7*, 1998.

[36] M. de Leoni *et al.*. Highly dynamic adaptation in process management systems through execution monitoring. *Business Process Management*, pages 182–197, 2007.

[37] P. Dadam *et al.*. A formal framework for workflow type and instance changes under correctness constraints. Technical report, 2003.

[38] C. A. Ellis *et al.*. A workflow change is a workflow. *Business Process Management, Models, Techniques, and Empirical Studies*, pages 201–217, 2000.

[39] C. A. Ellis *et al.*. The chautauqua workflow system. In *HICSS '97 : Proceedings of the 30th Hawaii International Conference on System Sciences*, 1997.

[40] S. Rinderle *et al.*. Disjoint and overlapping process changes : Challenges, solutions, applications. *On the Move to Meaningful Internet Systems 2004 : CoopIS, DOA, and ODBASE*, pages 101–120, 2004.

[41] R. Müller *et al.*. Rule-based dynamic modification of workflows in a medical domain. *Datenbanksysteme in Büro, Technik und Wissenschaft*, 1999.

[42] M. Reichert *et al.*. On the common support of workflow type and instance changes under correctness constraints. *On The Move to Meaningful Internet Systems 2003 : CoopIS, DOA, and ODBASE*, 2003.

[43] W. M. P. van der Aalst *etal.*. Identifying commonalities and differences in object life cycles using behavioral inheritance. *Applications and Theory of Petri Nets 2001*, pages 32–52, 2001.

[44] T. Basten. Branching bisimilarity is an equivalence indeed ! *Information Processing Letters*, 58(3) :141–147, 1996.

[45] G. Joeris *et al.*. Managing evolving workflow specifications. pages 310–319, 1998.

[46] M. Reichert *et al.*. Adaptive process management with adept2. *ICDE 2005. 21st International Conference on Data Engineering*, pages 1113–1114, 2005.

[47] S. Rinderle *et al.*. On representing instance changes in adaptive process management systems. In *WETICE '06 : Proceedings of the 15th IEEE International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises*, pages 297–304, 2006.

[48] R. Conradi *et al.*. Version models for software configuration management. *ACM Computing Surveys*, 30(2) :232–282, year = 1998,.

[49] R. Conradi *et al.*. Towards a uniform version model for software configuration management. In *System Configuration Management*, pages 1–17, 1997.

[50] W. Du *et al.*. Flexible specification of workflow compensation scopes. In *GROUP '97 : Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 309–316, 1997.

[51] Le CRM : un marché de 2,2 milliards d'euros en 2006. Journaldunet, 29/05/2007.

[52] E. Gamma *et al.*. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[53] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7) :558–565, 1978.

[54] A. Rashid *et al.*. A database evolution approach for object-oriented databases. In *ICSM*, pages 561–564, 2001.

[55] A. Rashid *et al.*. A database evolution taxonomy for object-oriented databases. *Journal of Software Maintenance and Evolution : Research and Practice*, 17(2) :93–141, 2005.

[56] J. Banerjee *et al.*. Semantics and implementation of schema evolution in object-oriented databases. *SIGMOD Rec.*, 16(3) :311–322, 1987.

[57] J. Banerjee *et al.*. Data model issues for object-oriented applications. *ACM Trans. Inf. Syst.*, 5(1) :3–26, 1987.

[58] R. Peters *et al.*. An axiomatic model of dynamic schema evolution in objectbase systems. *ACM Transactions on Database Systems*, 22 :75–114, 1997.

[59] A. H. Skarra *et al.*. The management of changing types in an object-oriented database. In *OOPLSA '86 : Conference proceedings on Object-oriented programming systems, languages and applications.* ACM, 1986.

[60] S. Monk *et al.*. Schema evolution in OODBs using class versioning. *SIGMOD Rec.*, 22 :16–22, September 1993.

[61] A. Björnerstedt *et al.*n. *Version control in an object-oriented architecture*. ACM, 1989.

[62] Y-G. Ra *et al.*. A transparent schema-evolution system based on object-oriented view technology. *IEEE Trans. on Knowl. and Data Eng.*, 9(4) :600–624, July 1997.

[63] W. Kim *et al.*. Versions of schema for object-oriented databases. In *VLDB '88 : Proceedings of the 14th International Conference on Very Large Data Bases*, pages 148–159. Morgan Kaufmann Publishers Inc., 1988.

[64] I. Santoyridis *et al.*. An object versioning system to support collaborative design within a concurrent engineering context. In *BNCOD 15 : Proceedings of the 15th British National Conferenc on Databases*, pages 184–199, London, UK, 1997. Springer-Verlag.

[65] R. H. Katz. Toward a unified framework for version modeling in engineering databases. *ACM Comput. Surv.*, 22(4) :375–409, 1990.

[66] J. Dehnert *et al.*. Bridging the gap between business models and workflow specifications. *Int. J. Cooperative Inf. Syst. 13 (2004) 289–332.*

[67] W. Gaaloul *et al.*. Mining workflow patterns through event-data analysis. In *SAINT-W'05 : Proceedings of the 2005 Symposium on Applications and the Internet Workshops*, pages 226–229, 2005.

[68] B. Weber *et al.*. Change patterns and change support features in process-aware information systems. *Advanced Information Systems Engineering*, pages 574–588, 2007.

[69] N. Russell *et al.*. Exception handling patterns in process-aware information systems. Technical report, BPMcenter.org, 2006.

[70] J. Price. Christopher alexander's pattern language. *Professional Communication, IEEE Transactions on*, 42(2) :117–122, 1999.

[71] D. Schmidt *et al.*. The design of the tao real-time object request broker. *Computer Communications*, 21(4) :294–324, October April 1998.

[72] B. Weber *et al.*. Change patterns and change support features in process-aware information systems, 2007.

Résumé

Un système de gestion de Workflows (*Workflow Management System* : WfMS) est un dispositif logiciel permettant de modéliser, d'automatiser et de surveiller l'exécution des flux de travail. Pour une automatisation efficiente, et mieux refléter le procédé réel, le modèle contenu dans le WfMS doit minimiser, tout en étant exempt d'erreurs, l'écart entre sa représentation et le processus réel. Pour ce faire, le déploiement du WfMS devrait éviter de reproduire la rigidité introduite par une planification rigoureuse des procédés métiers. En revanche, l'automatisation doit aussi admettre que des utilisateurs agréés puissent faire dévier les exécutions d'un Workflow de leur planification initiale, par exemple, face à une situation exceptionnelle, et faire évoluer les modèles de Workflows face aux optimisations ou à un changement des législations régissant les procédés métiers. Dans ce contexte, l'enjeu est d'offrir aux administrateurs du moteur d'exécution un support efficace pour la gestion du changement. La littérature propose une multitude de solutions *basées sur les états* d'avant et d'après le changement pour résoudre la problématique des modifications- par exemple, la cohérence et la robustesse - des Workflows en cours d'exécution. Sauf que la complexité accrue de ces solutions pose de nombreux soucis d'ingénierie lorsqu'il s'agit de les mettre en œuvre. Outre la difficulté de leurs exploitations, la traçabilité, le suivi, l'analyse et la réutilisation des changements sont considérablement limités. Dans ce mémoire, nous présentons une approche innovante basée sur des opérations pour la gestion des changements métiers (à savoir, l'évolution du schéma et la modification cohérente des instances en cours d'exécution) qui se veut être à un haut niveau de généricité sans, pour autant, affecter la robustesse des WfMS. Nous proposons pour cela un canevas formel *basé sur les opérations* et un nouveau paradigme de systèmes d'opérations- Stratégie/Incidence(s)- offrant un cadre flexible pour effectuer des changements, afin de faire évoluer les modèles de Workflow tout en préservant des exécutions correctes. Dans le but de préserver la correction des exécutions, par exemple, une incidence conformément à une stratégie, le critère de correction dynamique des opérations de changement que nous introduisons, permet de garantir (décidé localement dans un système d'opérations) qu'une Dérive (c'est-à-dire, l'incidence) puisse évoluer selon l'évolution de son Noyau (à savoir, la stratégie). Comme l'évolution des modèles ou les changements in situ des exécutions reflètent un savoir-faire et nécessitent une expertise, il est nécessaire de pouvoir capitaliser le changement en proposant des techniques pour leur réutilisation. Finalement, nous définissons un graphe des arrangements lexicographique d'opérations centré sur le noyau dont la topologie permet le suivi et la traçabilité des opérations de changement pour l'analyse et la réutilisation des modifications des modèles dans un WfMS.

Mots-clés: Morphflow, Flexibilité des Workflows, Système d'opérations, Gestion du changement, Stratégie/Incidence, Noyau/Dérive, Correction structurelle dynamique basée sur les opérations.